БИБЛИОТЕЧКА ПРОГРАММИСТА

Г. Л. СЕМАШКО
А. И. САЛТЫКОВ

Программирование на языке паскаль

БИБЛИОТЕЧКА ПРОГРАММИСТА

Г. Л. СЕМАШКО,
А. И. САЛТЫКОВ

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПАСКАЛЬ

Под редакцией
В. П. ШИРИКОВА

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1988

ББК 22.18
С30
УДК 519.6

Семашко Г. Л., Салтыков А. И. Программирование на языке паскаль. М.: Наука. Гл. ред. физ.-мат. лит., 1988.— 128 с.— (Библиотечка программиста.) — ISBN 5-02-013788-X.

Дается описание широко распространенного языка паскаль. Излагается в основном стандартный паскаль. Учитываются особенности работы на ЭВМ разных типов, более подробные сведения и указания приводятся для машин типа ЕС ЭВМ и БЭСМ-6.

Предлагаемая книга не требует от читателя специальной подготовки и имеет целью дать практические навыки, достаточные для самостоятельного составления несложных программ и запуска их на ЭВМ.

Для инженеров, научных работников, аспирантов, начинающих программировать на ЭВМ, а также для студентов, приступающих к изучению языка паскаль. Книга ориентирована на пользователей, работающих на машинах типа ЕС ЭВМ и БЭСМ-6.

Ил. 79. Библиогр. 14 назв.

С $\frac{1702070000-081}{053(02)-88}$ 42-88

ISBN 5-02-013788-X

ОГЛАВЛЕНИЕ

ПРЕДИСЛОВИЕ

Язык программирования паскаль, созданный Н. Виртом на рубеже 60—70-х годов, получил в настоящее время широкое распространение во всем мире. Трансляторы с паскаля имеются на отечественных ЭВМ многих типов, в том числе на БЭСМ-6 [8] и ЕС ЭВМ [13].

Важной особенностью паскаля, отличающей его от языков программирования, созданных ранее (фортрана, алгола и др.), является последовательное проведение в жизнь идей структурного программирования [12]. Другой существенной особенностью паскаля является концепция структуры данных как одного из фундаментальных понятий, лежащих, наряду с понятием алгоритма, в основе программирования [2].

К настоящему времени на русском языке издано немало книг по паскалю, в основном переводных [2, 3, 4, 5, 8, 10, 11]. Среди многочисленных книг, изданных на английском языке, отметим [14], отличающуюся полнотой и точностью изложения материала.

Эти книги ориентированы в основном на достаточно опытного читателя. В них рассматриваются, как правило, вопросы, не связанные с реализацией языка паскаль на ЭВМ конкретных типов. Между тем из опыта известно, что каждая конкретная реализация языка программирования содержит массу особенностей, представляющих трудности в первую очередь для начинающих пользователей ЭВМ.

При написании данной книги мы ставили себе задачу помочь пользователям машин типа БЭСМ-6 и ЕС ЭВМ, начинающим освоение языка паскаль. С этой целью материал книги разбит на две главы. Первая глава содержит наиболее простые и часто используемые конструкции языка паскаль, доступные пониманию начинающего читателя. Вторая глава содержит более сложный материал, адресованный достаточно опытному читателю.

Мы стремились к тому, чтобы читатель как можно раньше начал самостоятельно составлять программы. Поэтому во многих случаях материал излагается сначала на элементарном уровне

(и не в полном объеме). И лишь после того, как читатель «встанет на ноги» и приобретет начальный опыт, ему сообщаются более полные сведения по тому или иному вопросу.

Такая методика изложения материала, а также разбиение книги на две главы («элементарную» и более сложную) уже применялись нами ранее [9]. Наш опыт преподавания основ программирования в школах Дубны и в Дубненском филиале НИИЯФ МГУ убедил нас в правильности такого способа изложения материала.

В процессе работы над книгой нам пришлось экспериментально проверять «восприятие» трансляторами на БЭСМ-6 и ЕС ЭВМ тех или иных конструкций языка паскаль. Иногда обнаруживались расхождения между описаниями в [8] и [13] и фактическим положением вещей.

Считаем необходимым обратить внимание читателей на следующее обстоятельство.

Обычно в книгах тексты программ на паскале набираются строчными буквами с выделением ключевых слов полужирным шрифтом [1]. Однако пользователи ЭВМ получают с машины тексты, отпечатанные прописными буквами. Только прописными буквами выдаются тексты программ и на экраны большинства дисплеев. Поэтому в книге все тексты на языке паскаль набраны прописными буквами, более привычными для пользователей ЭВМ. Отметим, что такой способ набора принят и в [10].

Мы понимаем, что для достаточно полного и точного описания особенностей реализации языка паскаль на ЭВМ конкретного типа требуется несколько лет предварительной работы с соответствующим транслятором.

Данную книгу можно рассматривать как один из первых опытов описания языка паскаль для машин типа БЭСМ-6 и ЕС ЭВМ.

Появлению этой книги во многом способствовала Екатерина Ивановна Стечкина, которая много лет возглавляла редакцию, выпускавшую книги по информатике и вычислительной математике.

Выражаем искреннюю благодарность О. В. Благонравовой, Г. А. Косянину и Ю. К. Крюкову за помощь в подготовке примеров и задач, А. А. Корнейчуку и А. В. Гусеву за ценные советы и рекомендации, А. Н. Графовой за помощь в подготовке рукописи к изданию.

Мы будем признательны читателям, которые пришлют свои замечания и предложения.

Дубна, 1988 г. *Авторы*

ВВЕДЕНИЕ

Язык паскаль был создан в конце 60-х годов Н. Виртом как специальный язык для обучения студентов, но вскоре получил распространение среди программистов. Сейчас паскаль широко используется и для написания прикладных программ, и как язык системного программирования.

В частности, программное обеспечение многих мини- и микрокомпьютеров написано на языке паскаль. Трансляторы с этого языка имеются на наиболее распространенных типах ЭВМ во всем мире.

Основными причинами популярности языка паскаль являются следующие.

1. Простота языка позволяет быстро его освоить и создавать алгоритмически сложные программы.

2. Развитые средства представления структур данных обеспечивают удобство работы как с числовой, так и с символьной и битовой информацией.

3. Наличие специальных методик создания трансляторов с паскаля упростило их разработку и способствовало широкому распространению языка.

4. Оптимизирующие свойства трансляторов с паскаля позволяют создавать эффективные программы. Это послужило одной из причин использования паскаля в качестве языка системного программирования.

5. В языке паскаль реализованы идеи структурного программирования [2, 12], что делает программу наглядной и дает хорошие возможности для разработки и отладки.

В данной книге содержится информация для начинающих пользователей машин типа ЕС ЭВМ и БЭСМ-6, собирающихся работать на языке паскаль.

Описываемые возможности паскаля реализованы на машинах типа БЭСМ-6 и ЕС ЭВМ за исключением случаев, оговоренных

особо. Дополнительные возможности, предоставляемые трансляторами на конкретных ЭВМ, в книге не описаны. С ними можно ознакомиться в соответствующей литературе: [4, 8, 13]. Приведены только те режимы трансляции, которые имеются на ЭВМ перечисленных типов.

Учитывая, что многие читатели владеют языком фортран, мы будем в ряде случаев обращать их внимание на конкретные сходства и различия языков паскаль и фортран.

ГЛАВА I

ПАСКАЛЬ ДЛЯ НАЧИНАЮЩИХ

1. Задания для ЭВМ с программой на языке паскаль

Работа ЭВМ осуществляется под управлением специального комплекса программ, называемого *операционной системой* (*ОС*). Машины каждого типа могут быть оснащены несколькими операционными системами. Например, для машин серии ЕС ЭВМ разработаны операционные системы ДОС ЕС и ОС ЕС, для ЭВМ БЭСМ-6 существуют системы «Дубна», Диспак и другие.

Для того чтобы программа могла быть выполнена какой-либо ЭВМ, необходимо составить *задание* для этой ЭВМ.

В задание, помимо программы, входят так называемые *управляющие карты*. Программа, написанная на языке паскаль, может быть выполнена на любой ЭВМ, имеющей транслятор с этого языка, но управляющие карты должны быть свои для ЭВМ каждого типа.

Управляющие карты содержат обязательную информацию о задании. Например, машине надо «знать» фамилию программиста, шифр (пароль), под которым выполняется задание, и некоторые другие сведения.

Для каждой операционной системы имеется свой набор управляющих карт. Рассмотрим несколько примеров.

1.1. Задание для машин типа ЕС ЭВМ (вариант ОС ЕС)

Управляющие карты (или карты управления заданием) для машин типа ЕС ЭВМ начинаются с символа «/», располагаемого в первой колонке перфокарты (в первой позиции строки).

```
//XXXX␣JOB␣YYYY,'ФАМИЛИЯ',MSGLEVEL=(2,0)
//␣EXEC␣PASCLG
//PASC.SYSIN␣DD␣*
```

Программа на языке паскаль

```
//GO.SYSIN␣DD␣*
```

Данные для ввода

```
//
```

Здесь через XXXX и YYYY обозначены соответственно имя задания и шифр пользователя, а символом ␣ здесь и в дальнейшем будут обозначаться пробелы.

Пример 1. Пусть пользователь Петров, имеющий на одной из машин типа ЕС ЭВМ шифр PET12, собирается выполнить на этой ЭВМ задание под названием TASK1 (имя задания). Тогда первая карта задания будет такой:

```
//TASK1␣JOB␣PET12,PETROV,MSGLEVEL=(2,0)
```

Мы не будем утомлять читателя подробным описанием всех карт управления заданием, поясним только вторую карту.

```
//␣EXEC␣PASCLG
```

Здесь EXEC означает «выполнение» (EXECUTION), PAS — программа написана на языке паскаль, C — требуется трансляция (COMPILATION), L — работа редактора связей (LINKAGE EDITOR), G — выход на счет (GO).

Если выход на счет не требуется, то эта карта выглядит так:

```
//␣EXEC␣PASCL
```

На некоторых ЕС ЭВМ принят другой набор карт управления заданием. Приведем пример.

```
//XXXX␣JOB␣YYYY,'ФАМИЛИЯ',MSGLEVEL=(2,0)
//␣EXEC␣PASCLG
//C.SYSIN␣DD␣*
```

Программа на языке паскаль

```
//G.SYSIN␣DD␣*
```

Данные для ввода

```
//
```

Подробнее о картах управления заданием можно прочитать в [6].

1.2. Задание для ЭВМ БЭСМ-6 в системе «Дубна» (пакет задачи)

На ЭВМ БЭСМ-6 управляющие карты начинаются с символа '*' (в первой колонке).

```
*NAME␣XXXX
*PASS:YYYY
*TIME:00.05
*LIBRARY:11
*CALL␣ALLMEMORY
*PASCAL
```

Программа на языке паскаль

```
*EXECUTE
```

Данные для ввода

```
*END␣FILE
```

Здесь через XXXX и YYYY обозначены соответственно фамилия пользователя и его шифр. Если пакет задачи создан на перфокартах, то следует добавить еще одну перфокарту — так называемый «диспетчерский конец». Эта карта содержит все пробивки в 1-й и 41-й колонках, и притом только в этих колонках.

Пример 2. Пусть задание примера 1 должно быть выполнено на ЭВМ БЭСМ-6 и для его выполнения требуется не более двух минут времени. Первые три карты задания будут такими:

```
*NAME␣PETROV
*PASS:PET12
*TIME:00.02
```

Более подробно управляющие карты БЭСМ-6 описаны в [7, 9].

1.3. Задание в системе Диспак

Пример пакета выглядит так:

```
ШИФР␣5 1 5 3 1 0 3 С 2
Е Е В 1 А 3
```

```
ВРЕМ_0 5 3 0 0 0
*NAME_СЕМАШКО
*FICMEMORY
*PASCAL
*LIBRARY:11
```

Программа на языке паскаль

```
*EXECUTE
*END_FILE
```

«диспетчерский конец»
ЕКОНЕЦ

2. Словарь языка паскаль

Язык паскаль оперирует со следующим набором символов:
а) все латинские (и русские буквы на БЭСМ-6);
б) все арабские цифры;
в) ограничители и специальные символы:

+	:=	:	<	(	..
−	.	'	<=	)	{
*	,	=	>	[	}
/	;	<>	>=	]	↑

г) ключевые слова:

AND	END	NIL	SET
ARRAY	FILE	NOT	THEN
BEGIN	FOR	OF	TO
CASE	FUNCTION	OR	TYPE
CONST	GOTO	PACKED	UNTIL
DIV	IF	PROCEDURE	VAR
DO	IN	PROGRAM	WHILE
DOWNTO	LABEL	RECORD	WITH
ELSE	MOD	REPEAT	

В программе нельзя использовать идентификаторы, совпадающие по написанию с приведенными выше ключевыми словами.

Вместо символов { и }, отсутствующих на клавиатуре устройств подготовки данных, набираются соответственно пары символов (* и *).

Конструкция (* ТЕКСТ *) воспринимается как *комментарий* и может быть помещена в любом месте программы. На БЭСМ-6

нельзя ставить комментарий после END. (последнего оператора программы).

Пример. Пусть в каком-либо участке программы вычисляется корень уравнения. Тогда перед этим участком полезно поместить следующий комментарий:

(* ВЫЧИСЛЕНИЕ КОРНЯ УРАВНЕНИЯ *)

Пробелы, комментарии, концы строк являются *разделителями*. Между любыми именами, числами, ключевыми словами должен стоять по крайней мере один разделитель, а может их быть и сколько угодно. Но нельзя отделять один символ от другого внутри имени, числа либо ключевого слова.

Пример. Ключевое слово GOTO нельзя записать как GO TO (в отличие от фортрана, где это допускается).

Можно:	Нельзя:
GOTO 5005;	GOTO5005;
GOTO 5005;	GO TO 5005;
GOTO 5005;	GOTO 5 005;

На ЕС ЭВМ используется следующее представление символов:

Стандартное *)	[ ]	{ }	AND	OR	NOT	<>
Паскаль ЕС	(. .)	(* *)	&	!	¬	¬=

3. Данные

Программа, написанная на языке паскаль (равно как и программа, написанная на любом из других языков программирования), предназначена для обработки *данных*. Эти данные могут быть различной природы (числа, тексты, последовательности двоичных разрядов, или битов, и т. п.). Одни данные являются *исходными* (или, как говорят, задаются на входе), другие являются *результатами*, полученными из исходных данных в процессе выполнения программы (про такие данные обычно говорят, что они получаются на выходе).

В зависимости от способа их хранения и обработки в ЭВМ данные можно разбить и на две другие группы: *константы* и *переменные*.

Константы — это те данные, значения которых не изменяются в процессе работы программы. Значения переменных, в отличие

*) Используется в стандарте языка паскаль.

от констант, могут изменяться во время выполнения программы. Константы «узнаются» машиной по форме их записи, а переменные — по именам (или идентификаторам).

В языке паскаль используются константы трех видов: *числовые*, *булевские* и *символьные*. Первые предназначены для представления числовых данных (целых и вещественных). Булевские константы используются для представления данных, имеющих смысл логических высказываний (да — нет, истина — ложь). Символьные константы представляют данные, являющиеся последовательностями символов (тексты).

3.1. Константы

Числовые константы могут быть целыми и вещественными.

3.1.1. Целые константы (INTEGER). Целая десятичная константа представляет собой последовательность десятичных цифр, которой может предшествовать знак —.

Пример. 158; —15; 237845;

Целые константы в паскале не должны превосходить по абсолютной величине

на ЭВМ БЭСМ-6 $2^{40} - 1 \approx 10^{12}$,
на ЕС ЭВМ $2^{31} - 1 \approx 2 \cdot 10^9$.

На БЭСМ-6 допустимы и целые восьмеричные константы, которые представляют собой последовательности восьмеричных цифр, оканчивающиеся справа буквой B.

Пример. 135B; 2B;

Можно использовать и восьмеричные константы, оканчивающиеся вместо B буквами C либо T. Буква C означает, что последовательность восьмеричных цифр расположена в машинном слове *справа*, а слева дополняется нулями; буква T — последовательность восьмеричных цифр расположена в слове *слева*, а справа дополняется нулями.

Пример.

0025T есть константа 0025 0000 0000 0000,
146C есть константа 0000 0000 0000 0146.

3.1.2. Вещественные константы (REAL). Представление вещественных констант, как и на фортране, имеет две формы: в виде десятичной дроби, где вместо запятой используется точка (например, число 3.2), и в виде числа, содержащего указание на степень десяти (например, число 2.5E9).

Число, в записи которого использована степень десяти (например, $2{,}5 \cdot 10^9$), изображается на паскале так: знак умножения опус-

кается, вместо основания 10 пишется буква E, следом за E — показатель степени (т. е. так же, как и на фортране).

Пример.

2.5E9 соответствует числу $2{,}5 \cdot 10^{9}$,
0.13E—10 соответствует числу $0{,}13 \cdot 10^{-10}$.

Замечание. Константа, записанная в виде десятичной дроби, в отличие от констант фортрана, обязательно должна содержать как целую часть, так и дробную, т. е. нельзя записать 2. и .5, а следует использовать 2.0 и 0.5 .

Вещественные константы не должны по абсолютной величине превосходить

на БЭСМ-6 10^{19},
на ЕС ЭВМ 10^{76}.

Целое число может быть записано в виде вещественной константы, имеющей нулевую дробную часть (например, число 3.0).

Если вещественная константа по модулю меньше некоторого определенного числа, то машиной она воспринимается как нуль («машинный нуль»).

Для ЭВМ каждого типа эта наименьшая вещественная константа своя. Если обозначить ее через MINR, то

для БЭСМ-6 $MINR = 10^{-19}$,
для ЕС ЭВМ $MINR = 10^{-78}$.

Пример. Для БЭСМ-6 результат вычисления выражения $(10^{-10})^2$ есть машинный нуль, а для ЕС ЭВМ — число 10^{-20}.

Таким образом, ЭВМ каждого типа оперирует с конечным набором чисел из определенного диапазона.

Для БЭСМ-6 диапазон изменения вещественных чисел по модулю — от 10^{-19} до 10^{19}, для ЕС ЭВМ — от 10^{-78} до 10^{76}.

3.1.3. Булевские константы (BOOLEAN). Имеются две булевские константы: TRUE и FALSE.

3.1.4. Символьные константы (CHAR). Символ, заключенный в апострофы, есть символьная константа.

Пример. 'A', '1', '='.

3.1.5. Константы-строки. Последовательность символов, заключенная в апострофы, есть строка.

Пример. 'DUBNA'; '12345'; 'A*B';

Длиной строки K называется число символов в ней.

3.1.6. Константы типа ALFA*). Строку символов длиной в одно машинное слово называют константой типа ALFA.

---

*) Расширение стандарта языка паскаль для ЭВМ БЭСМ-6 и ЕС.

Длина К этой константы зависит от типа ЭВМ:

для БЭСМ-6 К = 6,
для ЕС ЭВМ К = 8.

Пример. Константы типа ALFA

на БЭСМ-6 'DUBNA_'; '141980'; 'МОСКВА';
на ЕС ЭВМ 'DUBNA_____'; 'IVANENKO';

Если среди символов константы-строки имеется апостроф, то он изображается двумя апострофами.

Пример. Константу A'B'C'D следует набрать как 'A''B''C''D'.

4. Идентификаторы

Идентификатор паскаля — это последовательность букв или цифр, начинающаяся с буквы. Значащими являются первые 8 символов. Заметим, что в паскале, в отличие от фортрана, пробелы в идентификаторах не допускаются.

Пример. A; B12, 15D13; STACKCOMP;

В последнем идентификаторе 9 символов. Последний, девятый символ не учитывается транслятором, так как значащими являются первые восемь символов. Но для мнемоники программисту иногда бывает удобно использовать более длинные идентификаторы.

Вместо слова «идентификатор» часто употребляют синонимы: «имя», «наименование», «название».

В программе нельзя использовать идентификаторы, совпадающие по написанию с приведенными в п.2 ключевыми словами.

5. О типах переменных

Каждая переменная (например A, B, C), используемая в программе, должна быть описана следующим образом:

A:TYPE1; C,B:TYPE2;...

Здесь A, B, C — идентификаторы переменных, TYPE1, TYPE2 — типы переменных.

Пример. J,K:INTEGER; A,B,C:REAL; L:BOOLEAN;

В языке паскаль имеется следующий набор типов переменных:

ПРОСТЫЕ ТИПЫ
  СКАЛЯРНЫЕ
    НЕСТАНДАРТНЫЕ (ПЕРЕЧИСЛЕНИЕ)
    СТАНДАРТНЫЕ
      ЦЕЛЫЙ (INTEGER)

ВЕЩЕСТВЕННЫЙ (REAL)
БУЛЕВСКИЙ (BOOLEAN)
СИМВОЛЬНЫЙ (CHAR)
СТРОКОВЫЙ (ALFA) *)
ОГРАНИЧЕННЫЕ (SUBRANGE)
СЛОЖНЫЕ ТИПЫ
МАССИВ (ARRAY)
МНОЖЕСТВО (SET)
ФАЙЛ (FILE)
ЗАПИСЬ (RECORD)
ССЫЛКИ (УКАЗАТЕЛЬ, POINTER)

Набор типов языка паскаль представлен на следующей схеме (рис. 1).

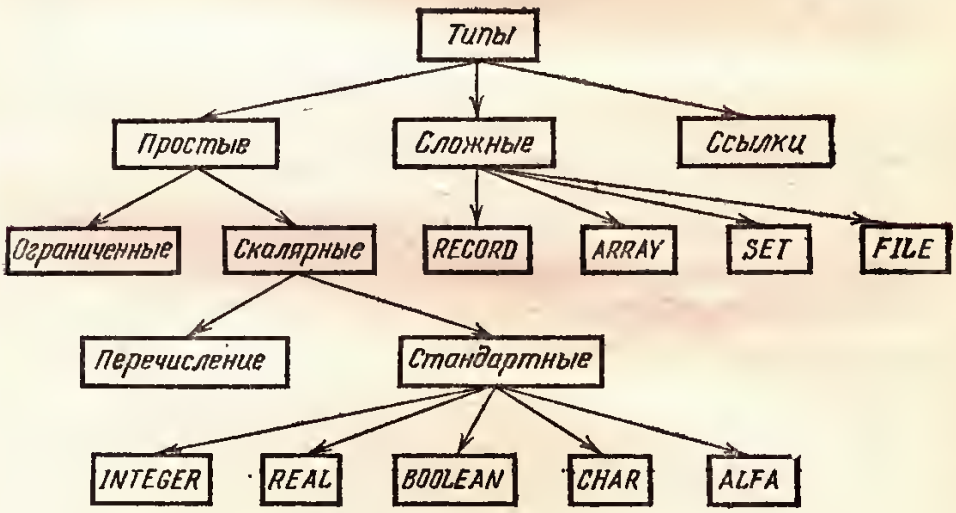


Рис. 1

Тип переменной определяет множество значений этой переменной, набор операций, которые к ней могут быть применены, а также тип результата выполнения этих операций.

Остановимся подробнее на типах, приведенных на схеме.

6. Скалярные типы

Каждый скалярный тип определяет соответствующее ему *упорядоченное* множество значений.

6.1. Тип целый (INTEGER)

Переменные типа INTEGER могут принимать только *целые* значения. Такие переменные описываются следующим образом:

```
A,B,C : INTEGER;
```

*) Расширение стандарта языка паскаль.

Здесь A,B,C...— имена переменных, INTEGER — тип переменных. Транслятор, встретив такое описание переменных A, B, C..., запоминает, что эти переменные могут принимать только целые значения и формирует соответственно этому команды программы.

Если используются операнды *целого* типа, то следующие операции дают результат *целого* типа:

* умножение.
DIV деление без округления — целая часть частного.
MOD остаток от деления A MOD B = A — ((A DIV B)*B).
+ сложение.
— вычитание.

Функции

ABS(X) — абсолютная величина X, SQR(X) — квадрат X.

Следующие функции дают *целый* результат и для X *вещественного*:

TRUNC(X) — (отбрасывание десятичных знаков после точки);
ROUND(X) — (округление до целого).

Операция (OP) над операндами целого типа выполняется правильно только при условии, что результат и каждый операнд по модулю не превосходят некоторой константы MAXINT.

$ABS(A\ OP\ B) \leqslant MAXINT,$
$ABS(A) \leqslant MAXINT,\ ABS(B) \leqslant MAXINT.$

для БЭСМ-6 $MAXINT = 2^{40}-1 \approx 10^{12}$
для ЕС ЭВМ $MAXINT = 2^{31}-1 \approx 2\cdot10^{9}$

Поясним на примерах работу приведенных операций и функций.

Пример 1. Пусть A = 14; B = 4. Тогда A DIV B дает 3; A MOD B дает 2 (остаток от деления); SQR(B) дает 16.

Пример 2. Пусть X = 8.915. Тогда TRUNC(X) дает 8; ROUND(X) дает 9.

Пример 3. Пусть надо на ЕС ЭВМ вычислить значение выражения $5\cdot10^{5}\cdot3\cdot10^{5}\cdot8\cdot10^{-5}$. Если запрограммировать так: 500000*300000*80E—5, то первое умножение не выполнится, так как результат $15\cdot10^{10}$ превышает MAXINT. Надо изменить порядок сомножителей, чтобы ЭВМ вычислила окончательный результат:

500000*80E—5*300000

Пример 4. Выражение 5*3 дает результат типа «целый», а 5*3.0 — типа «вещественный», так как один из сомножителей вещественный.

6.2. Тип вещественный (REAL)

Величины X этого типа могут принимать только вещественные значения.

Если хотя бы один из операндов вещественный, следующие операции дают вещественный результат: +, —, *,/.

Операция деления / дает вещественный результат и в случае двух целых операндов (в отличие от фортрана).

Пример. 6/2 дает 3.0

Следует обратить внимание, что стандартная функция ABS(X)— (модуль X) от целого аргумента дает целый результат, а от вещественного — вещественный, как и SQR(X) — квадрат X; но функции

SIN(X) — синус X (X в радианах),
COS(X) — косинус X (X в радианах),
LN(X) — натуральный логарифм X,
EXP(X) — экспонента X,
SQRT(X) — корень квадратный из X,
ARCTAN(X) — арктангенс X

дают вещественный результат как для вещественного, так и для целого аргумента.

Нельзя использовать переменные и константы типа REAL:

а) в функциях PRED(X), SUCC(X), ORD(X);

б) в качестве индексов массивов;

в) в операторах передачи управления в качестве меток;

г) в определении базы типа SET (см. п. 20).

6.3. Тип булевский (BOOLEAN)

Переменная булевского типа принимает значения TRUE или FALSE. Эти величины упорядочены следующим образом:

FALSE<TRUE

Операции AND, OR, NOT (применяемые к булевским операндам) дают булевские значения.

Операция AND (логическое умножение, пересечение, операция «И»). Выражение A AND B дает значение TRUE (истина), только в том случае, если и A и B имеют значения TRUE. Во всех остальных случаях значение выражения A AND B — FALSE (ложь).

⟨TRUE⟩ AND ⟨TRUE⟩ = ⟨TRUE⟩
⟨TRUE⟩ AND ⟨FALSE⟩ = ⟨FALSE⟩
⟨FALSE⟩ AND ⟨FALSE⟩ = ⟨FALSE⟩

Операция OR (логическое сложение, объединение, операция «ИЛИ»).

Выражение A OR B дает значение FALSE в том и только в том случае, если и A, и B имеют значения FALSE. Во всех остальных случаях результат — TRUE.

⟨TRUE⟩ OR ⟨TRUE⟩ = ⟨TRUE⟩
⟨TRUE⟩ OR ⟨FALSE⟩ = ⟨TRUE⟩
⟨FALSE⟩ OR ⟨FALSE⟩ = ⟨FALSE⟩

Операция NOT (отрицание, операция «НЕ»).

Выражение NOT A имеет значение, противоположное значению A.

NOT ⟨TRUE⟩ = ⟨FALSE⟩,
NOT ⟨FALSE⟩ = ⟨TRUE⟩.

Стандартными булевскими функциями являются: ODD(X), EOLN(X), EOF(X).

ODD(X) = TRUE, если X нечетный (X целый);
EOLN(X) = TRUE, если встретился конец строки текстового файла X;
EOF(X) = TRUE, если встретился конец файла X.

В остальных случаях эти функции принимают значение FALSE.

6.4. Тип символьный (CHAR)

Переменная типа CHAR может принимать значения из определенной упорядоченной последовательности символов, разрешенной транслятором с паскаля на данной ЭВМ.

Две стандартные функции позволяют поставить в соответствие данную последовательность символов множеству целых неотрицательных чисел (порядковым номерам символов последовательности).

Эти функции называются функциями преобразования:

ORD(C) — выдает номер символа C (нумерация с нуля),
CHR(I) — выдает I-й символ последовательности.

Пример. ORD(H) выдает номер символа H в последовательности всех символов, используемых транслятором. CHR(15) выдает 15-й символ этой последовательности, например букву O.

6.5. Тип ALFA*)

Переменная этого типа представляет собой машинное слово, содержащее символьную информацию.

Слово БЭСМ-6 содержит 6 символов,
Слово ЕС ЭВМ содержит 8 символов.

*) В стандарте языка паскаль этот тип отсутствует.

Переменная типа ALFA должна занимать *одно полное машинное слово*, т. е. содержать *ровно столько* символов, сколько их содержится в машинном слове.

К переменным этого типа можно применять операции отношения:

«равно» (=), «не равно» (<>), «меньше» (<), «больше» (>), «меньше или равно» (<=), «больше или равно» (>=).

Пример. Пусть A — переменная типа ALFA, содержащая слово 'IVANOV', B — содержит слово 'PETROV', а F — слово 'ROGOV_' (на БЭСМ-6). Тогда выражение

A=B принимает значение FALSE,
A<B (буква I предшествует P) — TRUE,
B>=F (буква P предшествует R) — FALSE.

6.6. Тип «перечисление»

В программу можно ввести и переменные какого-либо типа, не совпадающего ни с одним из стандартных. Такой тип задается перечислением значений, которые может принимать переменная.

Общий вид описания нестандартного типа:

TYPE NM=(WORD1, WORD2, ...,WORDN);

здесь NM — идентификатор типа (произвольный идентификатор), WORD1, WORD2... — конкретные значения, которые может принимать переменная типа NM. Эти значения считаются упорядоченными, т. е. описание типа одновременно вводит упорядочение

WORD1<WORD2<...<,WORDN.

Пример 1. TYPE COLOR=(RED, YELLOW, GREEN, BLUE);

Здесь определено, что RED<YELLOW<GREEN<BLUE. Переменная типа COLOR может принимать одно из перечисленных значений.

Ко всем переменным скалярного типа, кроме REAL, применимы следующие стандартные функции: SUCC(X), PRED(X), ORD(X).

Функция SUCC(X). По элементу X определяется та упорядоченная последовательность, которой принадлежит X, и выдается элемент, следующий за X в этой последовательности.

Пример 2. Пусть задана последовательность букв в алфавитном порядке. Тогда SUCC(A) есть B; SUCC(L) есть M и т. д. В примере 1 SUCC(RED) есть YELLOW.

Функция PRED(X). По элементу X определяется последовательность, которой принадлежит X, и выдается предыдущий элемент этой последовательности.

Пример 3. PRED(F) есть E; PRED(Z) есть Y и т. д.

Функция ORD(X). Выдается номер элемента X из соответствующей последовательности.

Пример 4. Если заданная последовательность есть латинский алфавит, то ORD(A) есть 0; ORD(C) есть 2. (Нумерация начинается с нуля).

Ко всем переменным одного и того же скалярного типа применимы операции отношения

=, < >, <=, >=, <, >.

7. Ограниченные типы (SUBRANGE)

Для переменной скалярного типа можно указать некоторое подмножество значений, которые может принимать данная переменная.

Общий вид:

A:MIN..MAX;

здесь A — переменная, MIN — левая граница, MAX — правая граница подмножества (диапазона). Границы диапазона разделяются двумя точками.

Тип MIN и MAX задает множество, определяющее основной тип переменной A (базовый тип). О переменной, описанной таким образом, говорят, что она имеет тип «ограниченный».

Пример. Пусть переменная K может принимать значения из множества 1 ÷ 20. Тогда ей приписывают ограниченный тип (SUBRANGE):K:1..20;— основным типом переменной K является тип INTEGER, так как границами диапазона являются целые константы 1 и 20.

Если переменная B может принимать одно из значений RED, YELLOW, GREEN, то эту переменную можно описать так: B:RED..GREEN; основным типом B является тип COLOR, описанный выше в примере 1. Граница MIN всегда должна быть меньше MAX.

Пример. Пусть I — переменная, принимающая значения года рождения сотрудника какого-либо учреждения. Очевидно, имеет смысл ограничить диапазон значений I подмножеством по крайней мере 1900, 1970, т. е. описать так: I:1900..1970; переменная I будет иметь тип ограниченный, а не целый.

8. Структура программы

Программа состоит из *заголовка* и *блока*. Проиллюстрируем структуру программы на следующем примере:

```
PROGRAM MA (INPUT, OUTPUT, F1, F2);
LABEL 15,120;
```

```
CONST INMAX=81;
    PI=3.14;
    OUTMAX=60;
    EOL='!';
TYPE FAMILY=(FATHER,MOTHER,CHILD);
    LLINE=1..INMAX;
      AR=ARRAY[1..16] OF CHAR;
VAR LINE:ARRAY[LLINE] OF CHAR;
    L:LLINE;
    I,K,POS:INTEGER;
    SP:AR; F:FAMILY;
    C,D:REAL;
PROCEDURE RLINE;
VAR M:LLINE; B:AR; Z:FAMILY;
    BEGIN
      M:=1;
      WHILE NOT EOLN (INPUT) DO;
        BEGIN READ (LINE[M]); M:=M+1
        END;
        READLN;
        LINE[M]:=EOL;POS:=1;
END; (*  КОНЕЦ ПРОЦЕДУРЫ  *)
BEGIN (*  НАЧАЛО РАБОТЫ ПРОГРАММЫ  *)
15:LINE[1]:=EOL;
120:RLINE;
 . . . . .
END.
```

8.1. Заголовок программы

В заголовке указывается имя программы и список параметров. Общий вид:

```
PROGRAM N (INPUT,OUTPUT,X,Y,...);
```

здесь N — имя программы; INPUT — файл ввода — указывается, если в программе есть ввод данных; OUTPUT — файл вывода — указывается всегда; X,Y — внешние файлы, используемые в программе (см. п. 21.1).

В приведенном выше примере заголовком является строка

```
PROGRAM MA (INPUT,OUTPUT,F1,F2);
```

8.2. Блок

Блок программы состоит из шести разделов, следующих в строго определенном порядке:

1) раздел меток (LABEL),
2) раздел констант (CONST),

3) раздел типов (TYPE),
4) раздел переменных (VAR),
5) раздел процедур и функций,
6) раздел действий (операторов).

Раздел действий должен присутствовать всегда, остальные разделы могут отсутствовать.

Каждый из первых 4-х разделов начинается с соответствующего ключевого слова (LABEL,CONST,TYPE,VAR), которое записывается один раз в начале раздела и отделяется от последующей информации только пробелом (либо концом строки либо комментарием).

В приведенном выше примере в блок входят строки от LABEL 15,120; до END.

8.2.1. Раздел меток (LABEL). Любой выполняемый оператор может быть снабжен меткой — целой положительной константой, содержащей не более 4-х цифр. Все метки, встречающиеся в программе, должны быть описаны в разделе LABEL.

Общий вид:

LABEL L1,L2,L3...;

здесь L1,L2,L3...— метки.

Пример. LABEL 5,10,100;

Метка отделяется от оператора двоеточием.

Пример. Пусть оператор A:=B; имеет метку 20. Тогда этот оператор выглядит так:

20:A:=B;

В приведенном выше примере программы в разделе LABEL описаны две метки 15 и 120, используемые в программе.

8.2.2. Раздел констант (CONST). Если в программе используются константы, имеющие достаточно громоздкую запись (например, число π с 8-ю знаками), либо сменные константы (например, для задания варианта программы), то такие константы обычно обозначаются какими-либо именами и описываются в разделе CONST, а в программе используются только имена констант. Это делает программу более наглядной и удобной при отладке и внесении изменений.

Общий вид:

CONST A1=C1; A2=C2; ...

здесь A1 — имя константы, C1 — значение константы.

Пример. CONST PI=3.14; C=2.7531;

В разделе CONST приведенной выше программы вводятся четыре константы, обозначаемые соответственно именами INMAX, PI, OUTMAX и EOL.

8.2.3. Раздел типов (TYPE). Если в программе вводится тип, отличный от стандартного, то этот тип описывается в разделе TYPE:

```
TYPE T1=<вид типа>;
     T2=<вид типа>;
     . . . . . .
```

где T1 и T2 — идентификаторы вводимых типов.

Пример. TYPE COLOR=(RED,YELLOW,GREEN,BLUE);

Здесь описан тип COLOR, задаваемый перечислением значений.

В приведенной выше программе вводятся типы FAMILY, LLINE и AR.

8.2.4. Раздел переменных (VAR). Пусть в программе встречаются переменные V11,V12,...; все они должны быть описаны следующим образом:

```
VAR V11,V12,...:TYPE1;
    V21,V22,...:TYPE2;...
```

здесь V11, V12, ... — имена переменных; TYPE1 — тип перемекных V11,V12,...; TYPE2 — тип переменных V21,V22,....

Пример. VAR K,I,J:INTECER; A,B:REAL;

Каждая переменная должна быть описана *до* ее использования в программе и отнесена к одному и только одному типу. Названия разделов (CONST,TYPE,VAR...) указываются только *один раз*.

Пример.

```
VAR A:REAL;
    B:REAL;
```

Таким образом, в разделе VAR вводится имя каждой переменной и указывается, к какому типу эта переменная принадлежит. Тип переменной можно задать двумя способами: указать имя типа (например, REAL,COLOR и т. д.) либо описать сам тип, например,

```
ARRAY[1..16] OF CHAR
```

Рассмотрим приведенную выше программу (см. п. 8). В разделе VAR описаны переменные с именами:

```
LINE,L,I,K,SP,C,D,F.
```

Для переменных

```
L,SP,I,K,C,D,F
```

указаны имена соответствующих типов (LLINE, AR, INTEGER, REAL, FAMILY). Часть из этих имен — стандартные (INTEGER, REAL), а типы LLINE, FAMILY и AR не являются стандартными. Эти типы должны быть описаны в разделе TYPE.

Тип переменной LINE никаким именем не назван и описан одновременно с описанием переменной.

Точно так же можно было бы поступить и для переменных L, SP, F:

```
L:1..INMAX;
SP:ARRAY[1..16] OF CHAR;
F:(FATHER,MOTHER,CHILD);
```

В этом случае в разделе TYPE эти типы не описываются. Но переменные M, B и Z (процедуры RLINE) имеют такие же типы, поэтому повторять громоздкие описания нерационально. Есть еще более существенная причина для описания типов LLINE, AR и FAMILY в разделе TYPE (см. п. 14.4), т. е. таких типов, к которым относятся как переменные PROGRAM, так и переменные процедуры.

О разделе процедур и функций речь пойдет ниже (см. п. 13 и 14).

8.2.5. Раздел действий (операторов). Эта часть программы начинается с ключевого слова BEGIN и заканчивается словом END, после которого должна стоять точка (END.). Раздел действий есть выполняемая часть программы, состоящая из операторов.

9. Операторы

Под операторами в языке паскаль подразумевают (в отличие от фортрана) только описание действий. Операторы отделяются друг от друга точкой с запятой. Если оператор стоит перед END, UNTIL или ELSE, то в этом случае точка с запятой не ставится.

9.1. Оператор присваивания

Общий вид:

```
V:=A;
```

здесь V — переменная, A — выражение, := — операция присваивания. Выражение A может содержать константы, переменные, названия функций, знаки операций и скобки.

Пример: `F:=3*C+2*SIN(X);`

Вид выражения однозначно определяет правила его вычисления: действия выполняются слева направо с соблюдением следующего старшинства (в порядке убывания):

1) NOT;
2) *, /, DIV, MOD, AND;
3) +, —, OR;
4) =, <>, <, >, <=, >=, IN.

Любое выражение в скобках вычисляется раньше, чем выполняется операция, предшествующая скобкам.

Присваивание допускается для переменных всех типов, за исключением типа файл.

В операторе V:=A переменная V и выражение A должны иметь один и тот же тип, а для типа SUBRANGE — одно и то же подмножество значений.

Замечание 1. Разрешается присваивать переменной типа REAL выражение типа INTEGER.

Замечание 2. В отличие от фортрана, нельзя присваивать переменной типа INTEGER выражение типа REAL.

9.2. Вывод информации на печать

Общий вид оператора:

WRITELN(P_1, P_2, ..., P_n);

здесь WRITELN — имя процедуры вывода (см. п. 13), P_1,P_2, ..., P_n — список выражений, значения которых выводятся на печать.

Кроме значений выражений, на печать можно выводить и произвольный набор символов. Для этого набор символов заключают в апострофы:

'⟨набор символов⟩'

Пример 1. WRITELN (' P=',P); Этот оператор выполняется так: сначала выводятся символы, заключенные в апострофы: P=. Затем выводится значение переменной P, например 13.5. На экране (на листинге) в результате работы оператора появится: P=13.5...

Пример 2. Вычислить длину окружности радиуса 5,782.

```
PROGRAM T10(OUTPUT);
CONST R=5.782;
VAR L:REAL;
  BEGIN
    L:=2*3.1416*R;
    WRITELN('␣L=',L)
  END.
```

либо

```
PROGRAM T11(OUTPUT);
VAR R:REAL;
  BEGIN R:=5.782;
    WRITELN('␣L=',2*3.1416*R)
  END.
```

Для оператора WRITELN не требуется оператор типа фортранного FORMAT, так как тип выражения (2*3.1416*R) либо тип переменной (L) определяют некоторую спецификацию, выбранную по умолчанию для переменных данного типа (REAL).

Первый символ строки на печать не выводится, а служит для управления устройством печати: если это 1, то устройство начнет печатать с начала новой страницы, если «пробел» — со следующей строки, если «+» — без перехода к новой строке, т. е. с наложением строк.

Если оператор вывода на печать составлен без учета роли первого символа в строке, то могут быть непредвиденные режимы печати: пропуск страниц, строк, наложение строк.

Замечание. Фортранная спецификация nX отсутствует в языке паскаль.

9.3. Примеры заданий для ЭВМ БЭСМ-6 и ЕС ЭВМ

1. Пусть программист Петров, имеющий шифр PET12, собирается выполнить программу примера 2 на ЭВМ БЭСМ-6 (ОС «Дубна»). Тогда ему надо составить следующее задание:

```
*NAME PETROV
*PASS:PET12
*TIME:00.05
*LIBRARY:11
*CALL ALLMEMORY
*PASCAL
  PROGRAM T13 (OUTPUT);
  VAR L:REAL;
    BEGIN
      L:=2*3.1416*5.782;
      WRITELN(' L=',L)
    END.
*EXECUTE
*END FILE
```

«Диспетчерский конец» (для колоды перфокарт).

2. Если же Петров назовет задание TEST1 и будет выполнять программу на ЕС ЭВМ, то задание будет выглядеть так:

```
//TEST1 JOB PET12, PETROV, MSGLEVEL=(2,0)
// EXEC PASCLG
//PASC.SYSIN DD—*
PROGRAM T13(OUTPUT);
VAR L:REAL;
  BEGIN
    L:=2*3.1416*5.782;
    WRITELN(' L=',L)
  END.
//GO.SYSIN DD—*
//
```

9.4. Оператор безусловного перехода GOTO

Общий вид:

```
GOTO N;
```

метка N, на которую передается управление, должна быть описана в разделе LABEL.

Пример.

```
PROGRAM T (OUTPUT);
LABEL  7;
VAR A,B:REAL;
  BEGIN
      . . . . . .
      GOTO 7
      . . . . .
    7:A:=B*3;
      . . . . . .
  END.
```

9.5. Составной оператор

Если при некотором условии надо выполнить определенную последовательность операторов, то их объединяют в один составной оператор.

Составной оператор начинается ключевым словом BEGIN и заканчивается словом END. Между этими словами помещаются составляющие операторы, которые выполняются в порядке их следования. После END ставится точка с запятой, а после BEGIN — только пробелы (либо комментарий).

Пример.

```
BEGIN
  I:=2;
  K:=I/5
END;
```

Слова BEGIN и END играют роль операторных скобок. Тело самой программы также имеет вид составного оператора. После последнего END программы ставится точка (END.). Нельзя извне составного оператора передавать управление внутрь его.

9.6. Оператор условного перехода

Этот оператор имсет две разновидности: IF и CASE.

9.6.1. Оператор IF. Булевские (логические) выражения могут принимать одно из двух значений: TRUE (истина) либо FALSE (ложь).

Простейшими логическими выражениями являются выражения отношения:

A1 OP A2

Здесь A1 и A2 — выражения, а OP — операция отношения. Операции отношения в языке паскаль обозначаются так:

= — равно;
> — больше;
< — меньше;
>= — больше или равно;
<= — меньше или равно;
<> — не равно.

Пример. 3<5; 18>=2; 5<=6; A=B;

Общий вид оператора IF:

IF A THEN ST;

здесь A — булевское выражение, ST — оператор (простой либо составной).

Если A — «истина», выполняется оператор ST. Если A — «ложь», то управление сразу передается следующему за ST оператору.

Пример. IF A <>0 THEN B:=X/A;

Если A≠0, то выполняется оператор B:=X/A; если A=0, то этот оператор пропускается и управление передается дальше, к следующему оператору.

Оператор IF может иметь и такой вид:

IF A THEN ST1 ELSE ST2;

здесь A — булевское выражение, ST1, ST2 — операторы.

Если A — «истина», выполняется оператор ST1; если A — «ложь» — выполняется оператор ST2, затем в обоих случаях управление передается к следующему оператору.

Замечание 1. Перед ELSE нельзя ставить точку с запятой.

Пример. IF A <>0 THEN B:=1/A ELSE B:=0; если A≠0, то переменной B присваивается значение 1/A; если A = 0, то — значение 0.

Замечание 2. Синтаксическая неоднозначность оператора

IF A1 THEN IF A2 THEN ST1 ELSE ST2;

трактуется так:

```
IF A1 THEN
  BEGIN IF A2 THEN ST1 ELSE ST2 END;
```

Пример.

```
IF A <>0 THEN IF B <>0
THEN C:=A/B  ELSE C:=0;
K:=-1;
```

Если A≠0, то для B≠0 C=A/B, а для B=0 C=0, затем выполняется оператор K:=—1. Если A = 0, то управление сразу передается на K:=—1.

9.6.2. Оператор CASE.

Общий вид:

```
CASE N OF
  M1,...,MN:ST1;
  K1,...,K1:ST2;
   . . . . . .
END;
A:=B;
```

здесь N — переключатель (селектор), MI,KI — метки (I = = 1, 2, ...), которые отличаются по смыслу от меток, описываемых в разделе LABEL. Переключатель и метки должны быть одного и того же скалярного типа, кроме REAL.

Оператор CASE передает управление тому оператору STI, с одной из меток которого совпало значение переключателя N, а затем — на следующий за END оператор.

Пример 1.

```
CASE I OF
     2:X:=0;
     3:X:=X*X;
   100:X:=SIN(X);
END;
     A:=B;
```

Если значение I есть 3, то выполняется оператор X:=X*X; а затем управление передастся на оператор A:=B.

Замечание. Метки оператора CASE не описываются в разделе LABEL и на них пельзя переходить оператором GOTO.

9.7. Операторы цикла

В языке паскаль имеется три вида операторов цикла:

WHILE, REPEAT и FOR

9.7.1. Оператор цикла WHILE.

Общий вид:

```
WHILE A DO ST
```

здесь A — булевское (логическое) выражение, ST — оператор (простой либо составной).

Выражение A вычисляется перед каждым выполнением оператора ST. Если A — TRUE, то ST выполняется и управление передается на вычисление A; если A — FALSE, то ST не выполняется и происходит выход из цикла. Если первоначальное значение A — FALSE, то ST не будет выполнен ни разу.

Пример.

```
WHILE X<>0 DO
  BEGIN C:=C+1/X;
    X:=X-1
  END;
```

В этом примере вычисляется выражение X <> 0. Если оно — «истина», (X≠0), будут выполняться операторы C:=C+1/X; X:=X-1 и управление опять будет передано на вычисление выражения X<>0.

Как только условие X ≠ 0 не выполняется, управление сразу передастся оператору, следующему за END; т. е. цикл повторяется пока X≠0.

Задача 1.

Вычислить сумму S=1+1/2+1/3+...+1/50

```
PROGRAM N1(OUTPUT);
VAR S:REAL; N:INTEGER;
BEGIN
    S:=0; N:=1;
    WHILE N<=50 DO
  BEGIN S:=S+1/N;
    N:=N+1
  END;
WRITELN('␣S=',S)
END.
```

Результат: S = 4.499...

9.7.2. Оператор цикла REPEAT.

Общий вид:

```
REPEAT STS UNTIL A;
```

здесь STS — группа выполняемых операторов, A — булевское выражение.

Работает оператор так: выполняются операторы STS, вычисляется выражение A; если оно «ложь» (FALSE), то вновь выполняются операторы STS, если A — «истина» (TRUE) — цикл заканчивается. Если A — «истина» с самого начала, то операторы STS выполняются один раз. Если A никогда не принимает значение

«истина», то группа операторов STS выполняется бесконечное число раз, происходит «зацикливание».

Пример. Рассмотрим такой цикл:

```
REPEAT C:=C+1/X; X:=X-1
UNTIL X = 0;
```

Сначала выполнятся операторы C:=C+1/X; X:=X—1, затем проверяется условие X = 0. Если X ≠ 0 (т. е. «ложь»), то повторяется выполнение указаиных операторов; если X = 0 («истина»), то управлеиие передается иа оператор, следующий за строкой UNTIL X = 0;

Задача 2. Решить задачу 1 с использованием оператора REPEAT

```
PROGRAM N2(OUTPUT);
VAR S:REAL;N:INTEGER;
  BEGIN
    S:=0; N:=1;
    REPEAT S:=S+1/N; N:=N+1;
  UNTIL N > 50;
  WRITELN('␣S=',S)
  END.
```

Результат S=4.499....

9.7.3. Оператор цикла FOR.

Общий вид:

```
FOR I:=N1 TO N2 DO ST;
```

здесь I — перемеиная цикла, N1 — иачальиое зиачеиие перемеиной цикла, N2 — коиечное зиачение, ST — оператор (простой либо составной).

I,N1 и N2 должны быть одиого и того же скаляриого типа, ио ие REAL. I принимает последовательиые значеиия данного типа от N1 до N2. Если N1 и N2 — целые числа, а I — целая переменная то шаг всегда равен едииице.

Пример. FOR I:=1 TO 20 DO A:=A+1;

Для I = 1, 2, 3, ..., 20 будет выполияться оператор A:=A+1; Если N1 и N2 символьного типа — например, соответственно, имеют значения A и Z, то переменная I принимает последовательиые значеиия в порядке алфавита: A, B, C, ..., Z.

Если N1 и N2 типа COLOR (тип COLOR=(RED,YELLOW, GREEN,BLUE)), иапример RED и GREEN соответственно, то переменная I принимает зиачеиия RED,YELLOW,GREEN.

Цикл по убызающим зиачениям параметра I от N2 до N1 имеет вид:

```
FOR I:=N2 DOWNTO N1 DO ST;
```

В этом случае параметр I принимает последовательные убывающие значения данного типа от N2 до N1.

Пример.

```
FOR I:=20 DOWNTO 1 DO A:=A+1;
```

I изменяется от 20 до 1 с шагом —1.

Задача 3. Вычислить сумму S = 1+1/2+1/3+. . .1/50

```
PROGRAM N3(OUTPUT);
VAR I:INTEGER; S:REAL;
  BEGIN S:=0;
    FOR I:=1 TO 50 DO
      S:=S+1/I;
    WRITELN (' S=',S)
  END
```

Оператор S:=S+1/I выполняется 50 раз соответственно для I = 1; 2; 3; . . . 50.

Результат S = 4.499 . . .

Замечания.

1. Внутри цикла нельзя изменять ни начальное, ни конечное значения переменной цикла, а также само значение переменной цикла.

2. Если в цикле по возрастающим значениям переменной начальное значение больше конечного, то цикл не выполняется ни разу.

Аналогично — для цикла с «DOWNTO», если начальное значение меньше конечного.

3. После завершения цикла значение переменной цикла «портится» (становится неопределенным).

Пример.

```
FOR I:=1 TO 50 DO
  BEGIN
    S:=S+1/I
    K:=I
  END;
L:=I;
```

В этом фрагменте K будет принимать последовательные значения 1, 2, 3, . . ., 50. После завершения всего цикла выполнится оператор L:=I. Будет ошибкой предполагать, что L содержит число 50: переменная I «испортилась» после окончания цикла. Если необходимо тем не менее запомнить последнее значение переменной I, то следует выполнить оператор L:=K.

10. Процедура ввода

Общий вид оператора:

READ (V1, V2, . . ., VN);

здесь V1, V2, . . ., VN — идентификаторы переменных.

Значения переменных пробиваются на картах (либо вводятся с терминала) и должны соответствовать типам переменных. Переменные V1, V2, . . ., VN могут быть одного из трех типов: INTEGER, CHAR (либо SUBRANGE этих типов) и REAL.

Пример.

```
PROGRAM N3(INPUT, OUTPUT);
VAR A, B, C : REAL; I:INTEGER;
  BEGIN
    READ (A,B,C,I);
    WRITELN ('␣A=',A,'␣B=',B,'␣C=',C,'␣I=',I);
  END.
```

Для ввода чисел 1.5, 2.15, —1.1, 25 их можно пробить так: 1.5␣2.15␣—1.1␣25

Если вводится последовательность символов, то пробел воспринимается как символ. В этом случае и конец строки (EOL) трактуется как символ «конец строки», а соответствующая переменная получает значение «пробел».

Пример. Пусть R:REAL; I:INTEGER; C1,C2,C3:CHAR; переменным R,C1,C2,C3,I надо присвоить соответственно значения 1.5, 'A', 'B', 'C', 25. На картах эти значения можно расположить одним из способов:

а) 1.5ABC25

б) +1.5E+0ABC
25

Но нельзя после 1.5 поместить пробел, так как он воспримется как значение символьной константы.

Оператор READ (R,C1,C2,C3,I); введет необходимые данные.

Конец входного файла и конец строки (карты) входного файла можно определить с помощью функций EOF и EOLN соответственно: функция EOF принимает значение TRUE только в случае исчерпания всех входных данных (конец файла); если исчерпались данные на одной карте (строке входного файла), то значение TRUE принимает другая функция — EOLN.

Если данные вводятся не с перфокарт или с терминала, а с какого-либо иного файла ввода F (магнитной ленты, диска), то используется оператор

READ (F,V1,V2, . . ., VN);.

Вводимые данные могут быть разделены одним или несколькими пробелами, но нельзя отделять пробелом знак числа либо одни цифры числа от других.

11. Процедура вывода

Процедура WRITE (P1,P2, . . . PN) дописывает в выходной файл OUTPUT значения выражений P1,P2, . . . PN.

Процедура WRITELN (P1,P2, . . . PN) дописывает в OUTPUT значения P1,P2,P3, . . . PN, заносит признак конца строки EOLN и выдает эту строку на печать.

Пример. Выдача на печать значений A, B может быть осуществлена по-разному:

а) WRITE(A);
WRITE(B);
WRITELN;

б) WRITE(A);
WRITELN(B);

в) WRITELN(A, B);

11.1. Формат вывода на печать

Транслятор отводит по умолчанию определенное число позиций для величин каждого стандартного типа.

Тип	CHAR	ALFA	BOOLEAN	INTEGER	REAL	
					M	N
Число позиций на БЭСМ-6	1	6	8	10	14	4
Число позиций на ЕС ЭВМ	1	8	5	12	24	16

Пример 1.

```
PROGRAM M (OUTPUT);
VAR I,J:INTEGER;
  BEGIN
    I:=2;
    J:=15;
    WRITELN ('␣I=',I'␣J=',J)
END.
```

Результат работы программы на БЭСМ-6 будет выглядеть так:

I=␣␣␣␣␣␣␣␣␣2␣J=␣␣␣␣␣␣␣␣15

на ЕС ЭВМ:

I=␣␣␣␣␣␣␣␣␣␣␣␣2␣J=␣␣␣␣␣␣␣␣␣␣␣15

Программист имеет возможность задать ширину поля (число позиций) M для выводимой величины P:

WRITE (P1:M1, P2:M2, . . .PN:MN);

Если MK избыточна, то поле [слева дополняется] пробелами; если MK недостаточна для размещения PK, то транслятор сам увеличивает ширину поля так, чтобы уместилось PK, а для БЭСМ-6 слева оставляет еще один пробел (кроме величин типа INTEGER, где пробел не предусмотрен).

Для вещественных значений можно задавать поля M и N, где M — общее число позиций, отводимых под все число PK, N — число позиций под его дробную часть.

П р и м е р: WRITE (P:10:2);

Здесь под P отводится 10 позиций, а 2 из них — под дробную часть.

П р и м е р. Оператор

(WRITELN (3.14159:0:2, 1=1, 3.14:7, 56, 'PAPA':5);

содержит пять выражений:

P1:3,14159
P2:1=1
P3:3.14
P4:56
P5:'PAPA'

Они изобразятся на листинге следующим образом:

а) на БЭСМ-6

3.14E+00 TRUE 3.1400E+00 56 PAPA

б) на ЕС ЭВМ этот оператор является ошибочным: нельзя указывать нулевую ширину поля и необходимо, чтобы было M > N.

12. Сложные типы переменных

Базируясь на простых типах, можно строить и более сложные типы переменных.

Переменные сложных типов состоят из отдельных компонент Тип компонент называют базовым типом данного сложного типа.

Таких сложных типов в языке паскаль четыре: массивы, записи, файлы и множества.

12.1. Массивы (ARRAY)

Массив — структура, состоящая из фиксированного числа компонент одного типа.

Общий вид описания массива (в разделе VAR):

A:ARRAY [TYPE1,TYPE2, . . .TYPEL] OF TYPEC;

здесь A — имя массива; TYPE1,TYPE2, . . . TYPEL — типы индексов; TYPEC — тип компонент (базовый тип).

Количество индексов (L) определяет размерность массива. Индексы могут быть любых скалярных типов, кроме REAL и INTEGER (не путать с SUBRANGE!). Индексы разделяются запятыми и заключаются в квадратные скобки.

П р и м е р 1. Пусть в памяти ЭВМ расположена таблица чисел, представляющая собой двумерный массив M:

15	10	2	30
7	19	55	11
22	18	6	3

Каждое число в таблице имеет тип INTEGER это — тип компонент (TYPEC).

Первый индекс — номер строки таблицы — может в данном примере быть от 1 до 3; второй индекс — номер столбца — может меняться от 1 до 4.

Таким образом, описание этого массива выглядит так:

M:ARRAY[1..3, 1..4] OF INTEGER;

Задав конкретные значения индексов, можно выбрать определенную компоненту массива. Например, оператор N:=M [1, 2] зашлет в N значение, стоящее в 1-й строке, 2-м столбце, т. е. 10.

12.1.1. Упакованные массивы. Если перед названием типа стоит ключевое слово PACKED, то при трансляции генерируется программа, плотно упаковывающая данные в ячейки памяти. Такая программа экономит память, но не время счета, так как для работы с элементами такого массива, как правило, требуется предварительная распаковка.

К упакованному массиву Z можно применить стандартные процедуры PACK и UNPACK.

Пусть массив B имеет тип ARRAY [M..N] OF TYPEC, а массив Z — PACKED ARRAY [U..V] OF TYPEC, где N—M $\geqslant$ V—U; Тогда

PACK(B,I,Z) означает «упаковку» элементов, начиная с I-й компоненты массива B, в массив Z.

UNPACK(Z,B,I) означает «распаковку», массива Z в массив B, начиная с I-го элемента массива B. Строка из N символов текстовой информации есть не что иное, как упакованный массив: PACKED ARRAY [1..N] OF CHAR;

Тип ALFA — частный случай упакованного массива:

PACKED ARRAY [1..8] OF CHAR на ЕС ЭВМ,
PACKED ARRAY [1..6] OF CHAR на БЭСМ-6.

13. Процедуры

При решении разных задач часто возникает необходимость проводить вычисления по одним и тем же алгоритмам, например, вычислять корень уравнения $f(x) = 0$. В языке паскаль предусмотрена возможность объединения любой последовательности операторов в самостоятельную подпрограмму, называемую процедурой. Процедуры, используемые во многих задачах, помещены в библиотеку процедур и находятся в памяти ЭВМ (оперативной либо внешней).

Те алгоритмы, которые программистом оформляются как процедуры в его собственной программе, должны начинаться с заголовка и кончаться оператором END;

Процедура паскаля — аналог фортранной подпрограммы.

Общий вид заголовка:

```
PROCEDURE N (P1:T1; P2:T2; VAR P3:T3, . . .);
```

здесь N — имя процедуры, PI — формальные параметры, TI — их типы.

Процедура имеет ту же структуру, что и главная программа (PROGRAM): разделы LABEL, CONST, TYPE, VAR и выполняемую часть (от BEGIN до END;).

Процедура помещается в главной программе после раздела VAR и перед BEGIN программы.

Пример.

```
PROGRAM MA (INPUT, OUTPUT);
VAR A:INTEGER;
    B:REAL; C:CHAR;
PROCEDURE N (P1:REAL; P2:CHAR);
  BEGIN (* НАЧАЛО РАБОТЫ ПРОЦЕДУРЫ *)
. . . . . .
  END; (* КОНЕЦ ПРОЦЕДУРЫ *)
BEGIN (* НАЧАЛО РАБОТЫ PROGRAM *)
  . . . . . .
END.
```

Формальные параметры — это наименования переменных, через которые передается информация из программы в процедуру либо из процедуры в программу.

Пусть, например, процедура SQ осуществляет решение квадратного уравнения $ax^2 + bx + c = 0$. Тогда она должна иметь пять

формальных параметров: для значений коэффициентов a, b, c и для результатов: x_1 и x_2.

Для того чтобы запустить процедуру в работу, необходимо к ней обратиться (ее вызвать). *Вызов* процедуры N производится оператором вида

```
N(P1,P2,P3, . . .);
```

Здесь N — имя процедуры, P1, P2, P3 — *фактические параметры*.

При вызове процедуры машина производит следующие действия. Устанавливает взаимно однозначное соответствие между фактическими и формальными параметрами, затем управление передает процедуре. После того, как процедура проработает, управление передается *вызывающей программе* на оператор, следующий за вызовом процедуры.

Соответствие между фактическими и формальными параметрами должно быть следующим:

а) число фактических параметров должно быть равно числу формальных параметров;

б) соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу.

Соответствующие параметры не обязательно должны быть одинаково обозначены.

Пример. Вызвать процедуру SQ можно так:

```
SQ(P, Q, R, Y, Z);
```

Здесь P,Q,R коэффициенты квадратного уравнения, а Y и Z — корни этого уравнения. Если вызвать SQ оператором SQ(X1,X2,A,B,C); то машина воспримет X1,X2,A как коэффициенты уравнения, а корни запишет в переменные B и C.

Пример. Составим процедуру SQ решения квадратного уравнения $ax^2 + bx + c = 0$ в предположении, что дискриминант неотрицателен.

```
PROCEDURE SQ(A,B,C:REAL; VAR X1, X2:REAL);
VAR D:REAL;
  BEGIN
    D:=B * B—4 * A * C;
    X1:=(—B+SQRT(D))/(2 * A);
    X2:=(—B—SQRT(D))/(2 * A)
  END;
```

С помощью этой процедуры решим квадратное уравнение $5{,}7y^2 - 1{,}2y - 8{,}3 = 0$

```
PROGRAM S (OUTPUT);
VAR Y1, Y2:REAL;
  PROCEDURE SQ (A,B,C:REAL; VAR X1, X2:REAL);
```

```
  VAR D:REAL;
BEGIN
    D:=B * B—4 * A * C;
    X1:=(—B+SQRT(D))/(2 * A);
    S2:=(—B—SQRT(D))/(2 * A)
  END;
BEGIN (* НАЧАЛО РАБОТЫ ПРОГРАММЫ *)
  SQ(5.7, —1.2, —8.3, Y1, Y2);
  WRITELN (' Y1=', Y1,' Y2=', Y2)
END.
```

Результат X1=0.49, X2=—5.2.

Как видно из примера, процедура помещается после декларативных операторов программы. Первым выполняется оператор обращения к процедуре

```
SQ(5.7, —1.2, —8.3, Y1, Y2);
```

Здесь первые три фактические параметра соответствуют формальным A, B, C, а последние два фактических параметра Y1 и Y2 соответствуют формальным X1 и X2. После того как процедура «запустится», в ячейки A, B, C попадут числа 5.7, —1.2, —8.3 и начнут выполняться операторы D:= . .., X1:=. . ., X2:=. . .

После окончания работы процедуры управление возвратится к оператору WRITELN, который отпечатает результат. Параметры процедур могут быть четырех видов: *параметры-значения*, *параметры-переменные*, *параметры-процедуры*, *параметры-функции*.

13.1. Параметры-значения

Если в качестве формального параметра указана переменная, то такой параметр и есть параметр-значение. Примерами таких параметров служат параметры A, B и C в процедуре SQ:

```
PROCEDURE SQ (A,B,C:REAL; VAR X1, X2:REAL);
```

В этом случае фактическим параметром, соответствующим A либо B либо C, может быть любое выражение соответственного типа, в частности, константа.

Например, обратиться к SQ можно так:

```
SQ((25./3+2)*2, —1.5, (8.2—3.1)/3, X1, X2);
```

Для параметров-значений машина при вызове процедур производит следующие действия: выделяет место в памяти для каждого формального параметра, вычисляет значение фактического параметра и засылает его в ячейку, соответствующую формальному параметру.

Если фактический параметр есть имя переменной, например, R, то значение этой переменной пересылается в соответствующий формальный параметр, например, A. На этом всякая связь между A и R обрывается.

Если даже фактический и формальный параметры одинаково обозначены, в памяти ЭВМ эти параметры занимают разные ячейки. Это полезно знать, чтобы не допустить распространенной среди начинающих программистов ошибки — пытаться передать информацию из процедуры в вызывающую программу через параметр-значение.

Пример.

```
PROGRAM T2 (OUTPUT);
VAR I:INTEGER; A:REAL;
  PROCEDURE P(I:INTEGER);
  BEGIN
  I:=I*2
  END;
BEGIN (* НАЧАЛО T2 *)
  I:=2;
  P(I);
  WRITELN ('␣ I=',I)
END.
```

В T2 происходит засылка числа 2 в ячейку, отведенную для переменной I, затем идет обращение к процедуре P с фактическим параметром I=2. При этом значение 2 пересылается в другую ячейку, отведенную для формального параметра I. В этой ячейке после выполнения оператора I:=I*2 появляется число 4. Но после возврата из процедуры на оператор WRITELN программа T2 «знает» только одну переменную I, которая по-прежнему содержит число 2. Поэтому программа напечатает I=2.

Если формальный параметр есть параметр-значение, то соответствующим фактическим параметром должно быть выражение того же типа, что и формальный параметр.

13.2. Параметры-переменные

Если перед именем формального параметра стоит ключевое слово VAR, то такой параметр есть параметр-переменная. Примерами таких параметров служат X1 и X2 в заголовке

```
PROCEDURE SQ (A,B,C:REAL; VAR X1, X2:REAL);
```

Фактический параметр, соответствующий параметру-переменной, может быть *только переменной* (не константой и не выражением).

При вызове процедур (функций) параметры-переменные обрабатываются так: для формального параметра используется именно та ячейка, которая содержит соответствующий фактический параметр.

Пример. При вызове процедуры SQ оператором SQ(P,Q, R,Y,Z) для переменных X1 и X2 используются непосредственно те ячейки, которые отведены для Y и Z. Поэтому оператор присваивания X1:=(—B+SQRT(D))/(2*A) засылает полученное значение в ячейку Y.

Под формальные и фактические параметры-значения транслятор отводит разные области памяти. Поэтому результат выполнения процедуры может быть передан только через параметр-переменную.

Пример.

```
PROGRAM L1351 (OUTPUT);
VAR A,B:INTEGER;
  PROCEDURE H (X:INTEGER; VAR Y:INTEGER);
  BEGIN
    X:=X+1; Y:=Y+1;
    WRITELN (X,Y)
  END;
BEGIN A:=0; B:=0;
  H(A,B);
  WRITELN (A,B)
END.
```

Результаты, выдаваемые процедурой H:1 и 1; программа печатает 0 и 1.

Разберем этот пример: фактический параметр A соответствует формальному параметру-значению X, а фактический параметр B — формальному параметру-переменной Y. Под параметры A и X отведены две *разные* ячейки памяти, а под B и Y — *одна и та же* ячейка.

При обращении к H(A,B) из ячейки A пересылается значение 0 в ячейку X, а в ячейку Y засылается адрес ячейки B, содержащей 0, так как в процедуре H параметр X — это параметр-значение, а Y — параметр-переменная.

При выполнении оператора X:=X+1 в ячейку X прибавляется 1 и в ячейке X окажется 1, а в ячейке A по-прежнему — 0.

Выполнение оператора Y:=Y+1 имеет следующий смысл: «взять число из ячейки, адрес которой находится в Y (т. е. из ячейки B), прибавить 1 и заслать в ту же ячейку (т. е. в B).»

Поэтому в результате выполнения оператора Y:=Y + 1 значение ячейки B станет 1. Оператор печати из процедуры WRITELN (X, Y) выдаст содержимое ячейки X и ячейки Y, т. е. 1 и 1. Оператор печати WRITELN (A, B) в программе напечатает содержимое A, которое осталось равным 0, и содержимое ячейки B, которое равно 1.

Процедуры в паскале допускают *рекурсию*, т. е. процедур может вызвать сама себя.

Если в процедуре P есть обращение к процедуре Q, описанной ниже, то перед описанием P процедура Q декларируется как FORWARD: после заголовка процедуры Q ставится двоеточие, а затем ключевое слово FORWARD *).

Пример.

```
PROCEDURE Q(X:T1):FORWARD;
PROCEDURE P(X:Y);
  BEGIN
    Q(A)
  END;
PROCEDURE Q;
  BEGIN
    P(B)
  END;
```

14. Функции

Функция в паскале является аналогом фортранной подпрограммы-функции и состоит из заголовка и блока.

Общий вид заголовка:

```
FUNCTION F(P1:T1; . . .PN:TN):TYPEF;
```

здесь F — имя функции; P1, . . .PN — формальные параметры; T1, . . .TN, . . . — их типы; TYPEF — тип результата.

Самостоятельный алгоритм можно оформить как функцию в том случае, если в качестве результата получается одно единственное значение.

К функциям обращение выглядит проще, чем к процедурам. Для вызова функции достаточно указать ее имя (с фактическими параметрами) в любом выражении.

Пусть, например, функция MAX (X,Y:REAL):REAL выдает значение бóльшего из параметров X,Y. Тогда оператор R:= =MAX(T,P*Q)*B/2 найдет бóльшее из значений T и P*Q, затем выполнит дальнейшие действия и зашлет результат в R.

После работы функции результат присваивается имени функции, поэтому в блоке функции обязательно должен присутствовать оператор присваивания вида:

⟨имя функции⟩:=⟨результат⟩;

---

*) Обращаем внимание, что в этом случае параметры процедуры описываются только в операторе с FORWARD. В заголовке самой процедуры параметры опускаются. Подробнее о рекурсии сказано в п. 14.3.

Пример. Вычисление большего из двух данных чисел можно оформить таким образом.

```
FUNCTION MAX (X,Y:REAL):REAL;
  BEGIN
    IF X>Y THEN MAX:=X ELSE MAX:=Y
  END;
```

Процедуры, функции, программы часто называют модулями.

14.1. Побочные эффекты

1. Если в теле процедуры (функции) перевычисляется некоторая *нелокальная* переменная, т. е. такая переменная, которая описана в других модулях, содержащих данный, то могут наблюдаться непредвиденные последствия.

Пример. Пусть функция F (X) имеет такой вид:

```
FUNCTION F(X:REAL):REAL;
  BEGIN
    V:=V*X;
    F:=SQRT(V)+X
  END;
```

т. е. в процессе работы функция F изменяет некоторую нелокальную величину V. Рассмотрим теперь два выражения, которые вычисляются в программе, содержащей F(X):

F(X)+V и V+F(X).

Эти выражения дадут *разные* результаты, так как в первом случае к F(X) прибавится уже *измененное* значение V (в процессе работы F), а во втором случае к *первоначальному* значению V добавляется F(X).

2. Вторая опасность заключается в неправильном использовании параметров-переменных в качестве формальных параметров.

Пример. Найти 5-й член последовательности

$a_{n+1} = 3a_n - 2,$

$a_1 = 1.$

Опасно оформлять функцию в виде

```
FUNCTION F (VAR A,N:INTEGER):INTEGER;
VAR I:INTEGER;
  BEGIN
    FOR I:=1 TO N DO A:=3*A-2;
    F:=A
  END;
```

Так, если обратиться к этой функции оператором B:=F(1,5), будет «испорчена» константа 1, так как в ячейку памяти (первый фактический параметр), содержавшую ранее единицу, функция F поместит

текущий член последовательности, и при дальнейшей работе программы вместо 1 будет использоваться значение a_5. Такие ошибки бывает трудно найти, поэтому полезно придерживаться следующего правила: в функциях не использовать параметры-переменные.

14.2. Параметры-процедуры. Параметры-функции

Такие параметры в списке формальных параметров предваряются ключевыми словами PROCEDURE и FUNCTION соответственно.

Пример 1. PROCEDURE P(PROCEDURE A); Здесь процедура P имеет один параметр-процедуру A.

Пример 2.

PROCEDURE Q (FUNCTION S:REAL; B:REAL);

Процедура Q имеет два параметра: параметр-функцию S и параметр-значение B.

Замечание 1. На ЕС ЭВМ имеются *отличия от стандарта* в описании параметров-функций (процедур). Транслятор требует перечислить все параметры функции, являющейся параметром.

Пример 3. PROCEDURE Q (FUNCTION F(I:INTEGER): REAL); Здесь формальный параметр F — функция от одного целого аргумента, результат F — вещественный.

Если вызывается процедура (функция), имеющая параметр-процедуру (функцию), то соответствующий фактический параметр должен совпадать по типу результата с формальной процедурой (функцией). Программисту необходимо внимательно следить за совпадением типов результатов, так как в случае нарушения этого правила *никакой диагностики не выдается*, а программа работает неверно. Поясним на примере.

Обратиться к процедуре Q(FUNCTION F(I:INTEGER):REAL) можно так: Q(SINUS(K)); где SINUS(K) есть SIN(K). Если K имеет тип INTEGER, тогда SINUS(K) — типа REAL. Это совпадает с типами I и F в заголовке Q. Нельзя, однако обратиться к Q с функцией ABS(K), а именно: Q(ABS(K)); в этом случае тип формального параметра F—REAL, а тип фактического ABS(K) — INTEGER, т. е. формальный и фактический параметры не совпадают по типу.

Задача. Составить процедуру выдачи таблицы произвольной вещественной функции. Процедура должна иметь следующие формальные параметры: вещественную функцию, нижнюю границу аргумента, верхнюю границу аргумента, шаг по аргументу.

```
PROCEDURE TAB (FUNCTION F:REAL;
                        LOW, UP, STEP:REAL);
VAR X:REAL;
    J:INTEGER;
```

```
BEGIN
  X:=LOW;
  FOR J:=0 TO TRUNC ((UP—LOW)/STEP) DO
    BEGIN
      WRITELN (X:10,F(X):10);
      X:=X+STEP
    END
END;
```

(см. Замечание 2)

Выражение TRUNC ((UP—LOW)/STEP) дает число точек, в которых вычисляется функция F (при счете от 0).

Если к процедуре TAB обратиться оператором

```
TAB (SIN, 0.0, 6.4, 0.33);
```

то будет напечатана таблица функции sin x для x от 0 до 6.4 с шагом 0.33. Алгоритмы, употребляемые наиболее часто различными пользователями, оформляются в виде процедур и функций, помещаются в память машины и составляют библиотеку стандартных программ (модулей).

З а м е ч а н и е 2. Многие трансляторы, в том числе на БЭСМ-6 и ЕС ЭВМ, *не допускают* использования стандартных функций в качестве фактических параметров. Для таких трансляторов оператор

```
TAB (SIN, 0.0, 6.4, 0.33);
```

является ошибочным, так как SIN(X) — стандартная функция. Это ограничение можно легко обойти, введя новую функцию, эквивалентную стандартной.

П р и м е р. Введем функцию SINUS(X) таким образом:

```
FUNCTION SINUS (X:REAL):REAL;
BEGIN
  SINUS:=SIN(X)
END;
```

Тогда составить таблицу sin x можно, обратившись к процедуре TAB следующим оператором: TAB(SINUS, 0.0, 6.4, 0.33);

При использовании параметров-процедур и параметров-функций надо иметь в виду возможные осложнения.

1. Ошибки, допускаемые программистом в процедурах, имеющих параметры-процедуры и параметры-функции, иногда бывает трудно найти, что ведет к длительной отладке таких процедур.

2. Если число и тип параметров формального параметра-функции не совпадает с числом либо типом параметров соответствующего фактического параметра-функции, то такая программа не может быть правильно выполнена, а многие версии трансляторов с паскаля ве выдают в этом случае никакой диагностики.

3. Правила языка паскаль требуют, чтобы фактические параметры-процедуры (функции) содержали только параметры-значения. Это накладывает серьезные ограничения на использование параметров-процедур и параметров-функций.

14.3. Рекурсии

В языке паскаль процедуры и функции могут вызывать сами себя, т. е. обладать свойством рекурсивности.

Пример. Выдать на печать в обратном порядке цифры целого положительного числа N. Составим процедуру REVERSE

```
PROCEDURE REVERSE (N:INTEGER);
BEGIN
  WRITE (N MOD 10);
  IF (N DIV 10)< > 0
  THEN REVERSE (N DIV 10)
END;
```

Рассмотрим работу этой процедуры для числа N = 153. Оператор WRITE (N MOD 10) выдает в файл OUTPUT остаток от деления числа 153 на 10, т. е. последнюю цифру 3. Оператор

```
IF (N DIV 10) < > 0
THEN REVERSE (N DIV 10)
```

проверяет целую часть частного 153/10 = 15 на ноль. Если целая часть не ноль, то с этим значением (15) идет вновь обращение к процедуре REVERSE. Оператор WRITE (N MOD 10) дописывает в файл OUTPUT остаток от деления 15 на 10, т. е. 5; затем со значением 15/10 = 1 идет обращение к REVERSE. После вывода цифры 1 оператором WRITE (N MOD 10) проба N DIV 10 на ноль передает управление на конец процедуры. В файле OUTPUT будет записано число 351. Его можно выдать оператором WRITELN.

Таким образом, *однократное* обращение извне к процедуре REVERSE вызвало *трехкратное* ее срабатывание. Условие полного окончания работы рекурсивной процедуры должно находиться *в самой процедуре* (иначе произойдет зацикливание).

Рекурсивные процедуры и функции (модули) имеют одну из двух форм: *прямую рекурсию* и *косвенную рекурсию*. В первом случае модуль содержит оператор вызова *этого же модуля*, как в приведенной выше процедуре REVERSE. Во втором случае один модуль вызывает какой-либо другой модуль, который либо сам либо посредством других модулей вызывает исходный модуль.

Пример. Если A, B — имена модулей, то схема вызова может быть такой: A → B → A.

В случае косвенной рекурсии возникает проблема: как и где описать вызываемый модуль. По правилам языка паскаль каждый

вызываемый модуль должен быть описан *до его вызова*. Но если модуль A вызывает B, а B вызывает A, то получается замкнутый круг. Для подобных ситуаций принято следующее правило: один из рекурсивных модулей, вызывающих друг друга, описывается *предварительно* следующим образом:

PROCEDURE P (⟨список форм. параметров⟩); FORWARD;

здесь P — имя процедуры, FORWARD — ключевое слово. Это описание указывает транслятору, что текст процедуры P помещен ниже.

Подобным же образом описывается функция: к оператору FUNCTION добавляется слово FORWARD. Список формальных параметров и тип результата (для FUNCTION) включается только в это предварительное описание и опускается в заголовке соответствующего модуля.

П р и м е р. Пусть функция B при работе вызывает функцию A, которая, в свою очередь, вызывает функцию B. Тогда эти модули можно описать так:

```
FUNCTION B(X:INTEGER):REAL; FORWARD;
FUNCTION A(Y:INTEGER):REAL;
  BEGIN
    ...
    A:=B(I) + 3.5
  END;
FUNCTION B;
  BEGIN
    ...
    B=A(D)—1.8
  END;
```

Заголовок (FUNCTION B) перед текстом функции B содержит только имя этой функции (список формальных параметров и тип функции не указываются).

14.4. Локальные и глобальные переменные

Напомним, что каждый модуль (процедура, функция, программа) состоит из заголовка (PROCEDURE..., FUNCTION..., PROGRAM...) и блока.

П р и м е р 1.

```
PROCEDURE P(A:REAL; VAR X:REAL); (* ЗАГОЛОВОК *)
VAR K:INTEGER; (* НАЧАЛО БЛОКА *)
  BEGIN
    ...
  END; (* КОНЕЦ БЛОКА *)
```

Вложенные процедуры. Если блок какой-либо процедуры P1 содержит внутри процедуру P2, то говорят, что P2 *вложена* в P1.

Пример.

```
PROCEDURE P1 (X:REAL; VAR Y:REAL);
VAR C:INTEGER;
  PROCEDURE P2 (VAR Z:REAL);
    . . . . . .
  END;
BEGIN
  ...
END;
```

Любые идентификаторы, введенные внутри какого-либо блока (процедуры, функции) для описания переменных, констант, типов, процедур, называются *локальными* для данного блока. Такой блок вместе с вложенными в него модулями называют *областью действия* этих локальных переменных, констант, типов и процедур.

Пример 2.

```
PROCEDURE T1;
VAR Y1, Y2:REAL;
  PROCEDURE SQ1;
  VAR A,B,C,D:REAL;
  BEGIN
  (* ПЕРЕМЕННЫЕ A, B, C, D ЯВЛЯЮТСЯ ЛОКАЛЬНЫ-
  МИ ДЛЯ SQ1, ОБЛАСТЬ ИХ ДЕЙСТВИЯ — ПРОЦЕДУРА
  SQ1 *)
  . . . . . .
  END;
BEGIN
(* ПЕРЕМЕННЫЕ Y1, Y2 — НЕЛОКАЛЬНЫЕ ДЛЯ SQ1 —
ОБЛАСТЬ ИХ ДЕЙСТВИЯ — T1 И SQ1 *)
. . . . . .
END;
```

Константы, переменные, типы, описанные в блоке PROGRAM, называют *глобальными*. Казалось бы, проще иметь дело вообще только с глобальными переменными, описав их все в PROGRAM. Но использование локальных переменных позволяет системе лучше оптимизировать программы, делает их более наглядными и уменьшает вероятность появления ошибок.

При написании программ, имеющих вложенные модули, необходимо придерживаться следующих правил:

1. Описывать идентификаторы в том блоке, где они используются, если это возможно.

2. Если один и тот же объект (переменная, тип, константа) используется в двух и более блоках, то описать этот объект надо в самом внешнем из них, содержащем все остальные блоки, использующие данный объект.

3. Если переменная, используемая в процедуре, должна сохранить свое значение до следующего вызова этой процедуры, то такую переменную надо описать во внешнем блоке, содержащем данную процедуру.

Локализация переменных дает программисту большую свободу в выборе идентификаторов. Так, если две процедуры A и B полностью отделены друг от друга (т. е. не вложены одна в другую), то идентификаторы в них могут быть выбраны совершенно произвольно, в частности, могут повторяться. В этом случае совпадающим идентификаторам соответствуют разные области памяти, совершенно друг с другом не связанные.

Пример 3.

```
PROGRAM T2 (OUTPUT); VAR K:INTEGER;
  PROCEDURE A;
  VAR X,Z:REAL;
  BEGIN (* НАЧАЛО A *)
  (* ЧЕРЕЗ X, Z ОБОЗНАЧЕНЫ ДВЕ ВЕЛИЧИНЫ —
  ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ ДЛЯ A;
  K — ГЛОБАЛЬНАЯ ПЕРЕМЕННАЯ ДЛЯ A *)
  . . . . . .
  END; (* КОНЕЦ A *)
  PROCEDURE B;
  VAR X,Y:INTEGER;
  BEGIN (* НАЧАЛО B *)
  (* ЧЕРЕЗ X, Y ОБОЗНАЧЕНЫ ДВЕ ДРУГИЕ ВЕЛИЧИ-
  НЫ — ЛОКАЛЬНЫЕ ПЕРЕМЕННЫЕ ДЛЯ B; K —
  ГЛОБАЛЬНАЯ ПЕРЕМЕННАЯ ДЛЯ B *),
  . . . . . .
  END; (* КОНЕЦ B *)
BEGIN (* НАЧАЛО РАБОТЫ PROGRAM T2 *)
(* K — ЕДИНСТВЕННАЯ ПЕРЕМЕННАЯ, КОТОРУЮ
МОЖНО ИСПОЛЬЗОВАТЬ В T2 *)
. . . . . .
END.
```

Если один и тот же идентификатор описан в блоке B и второй раз описан во вложенном в B блоке C, то надо помнить, что эти два одинаковых идентификатора соответствуют *разным* ячейкам памяти.

Пример 4.

```
PROGRAM T3 (OUTPUT);
VAR I:INTEGER; A:REAL;
```

```
  PROCEDURE P(VAR D:REAL);
  VAR I:INTEGER;
  BEGIN (* НАЧАЛО P *)
    I:=3;
    D:=I+10*D
  END; (* конец P *)
BEGIN (* НАЧАЛО T3 *)
  A:=2.0;
  I:=15;
  P(A);
  WRITELN ('␣I=',I, '␣A=',A)
END.
```

Глобальным переменным I и A отводятся две ячейки памяти. Первыми выполняются операторы A:=2.0 и I:=15. Затем вызывается процедура P(A). В процессе работы P отводится ячейка для локальной переменной I и туда засылается число 3. После окончания работы процедуры P эта ячейка I программой «забывается». После возврата на оператор WRITELN программа «знает» только одну ячейку I — глобальную, т. е. ту, которая содержит число 15. Поэтому программа T3 выдаст на печать I = 15, A = 23.0, так как A = 3 + 10*2.

Если локальная и глобальная переменная принадлежит к одному и тому же сложному типу, то этот тип надо описать в разделе TYPE, а сами переменные описывать через этот общий тип.

Пример.

```
PROGRAM T1 (OUTPUT);
TYPE AB=ARRAY [1..3] OF REAL;
VAR A:AB;
  PROCEDURE Q;
  VAR B:AB;
  . . . . . .
END.
```

В этом примере переменные A и B описаны через общий тип AB. Если же локальная и глобальная переменные описаны одинаково, но не через общий тип, то программа может «не понять», что эти переменные принадлежат одному типу.

Пример.

```
PROGRAM T2 (OUTPUT);
VAR A:ARRAY [1..3] OF REAL;
  PROCEDURE Q;
  VAR B:ARRAY [1..3] OF REAL;
  . . . . . .
END.
```

В этом примере переменные A и B — одинаковые массивы, т. е. типы этих переменных одинаковы, но программа тем не менее «не считает», что A и B — принадлежат одному типу. Это происходит из-за того, что описание массивов дано в разных блоках.

15. Стандартные процедуры и функции

Наиболее часто употребляемые библиотечные процедуры и функции называют стандартными. Для работы с ними не надо ни заказывать библиотеку, ни описывать их предварительно в программе.

Часть приводимых ниже процедур (от RESET до DISPOSE) будут описаны во II части книги, поэтому при первом чтении их можно опустить.

1) ORD(C) — выдает порядковый номер символа C в упорядоченной последовательности символов, задаваемой транслятором. Нумерация начинается с нуля.

2) CHR(I) — выдает символ с порядковым номером I. Функции ORD(C) и CHR(I) взаимнообратны, т. е. CHR(ORD(C))=C и ORD(CHR(I))=I. В силу упорядоченности последовательности символов, ORD(C1) < ORD(C2), если C1 < C2 и вообще, если R — логическая операция отношения (=, <>, >, <,>=, <=), то из ORD(C1) R ORD(C2) следует C1 R C2 и обратно.

3) PRED(C) — выдает значение, предшествующее C в упорядоченной последовательности значений.

4) SUCC(C) — выдает значение, следующее за C. Результат этих двух функций будет неопределенным, если соответствующих значений не существует; в этом случае не все трансляторы выдают диагностику.

5) ODD(X) — выдает TRUE, если X нечетно (X — целое).

6) элементарные математические функции:

ABS(X) — модуль X,
SQR(X) квадрат числа X

(обратить внимание: нельзя на БЭСМ-6 использовать выражение X**2). Эти две функции выдают результат того же типа, что и аргумент.

Замечание. В этом, в общем удобном для программиста свойстве, кроется и источник «тяжелых» ошибок.

Например, одна из этих функций является фактическим параметром. Смена типа аргумента приведет к смене типа фактического параметра. Но тип соответствующего формального параметра строго фиксирован в описании соответствующего модуля, т. е. смена начальных данных в такой программе может привести к ошибке, которой при других данных не было.

TRUNC(X) — целая часть X, получающаяся при *отбрасывании* знаков после десятичной точки. Результат — целый, X — вещественный.

Пример.

TRUNC (3.9) дает 3,
TRUNC (—3.9) дает —3.

ROUND (X) — вещественный X *округляется* до целого, выдаваемого как результат.

Пример.

ROUND(3.9) дает 4,
ROUND(—3.9) дает —4.

Аргумент следующих функций может быть как вещественным, так и целым. Результат — всегда вещественный.

LN(X) — натуральный логарифм от X,
EXP(X) — *e* в степени X,
SQRT(X) — квадратный корень из X,
SIN(X) — синус X,
COS(X) — косинус X,
ARCTAN(X) — арктангенс X.

7) PACK(A, K, Z) — осуществляет упаковку *одномерного* массива A, начиная с K-й компоненты, в массив Z с 1-й компоненты. (Массив Z типа PACKED).

8) UNPACK (Z, A, K) — осуществляет распаковку *одномерного* массива Z начиная с 1-й компоненты в массив A с K-й компоненты. (Массив Z типа PACKED).

9) RESET(F) — устанавливает указатель файла на первую компоненту файла F и считывает ее в «окно» *) F↑; функции EOF(F) присваивается значение FALSE, если файл не пуст; иначе — значение F↑ не определено и функция EOF(F) имеет значение TRUE.

10) REWRITE(F) — очищает файл (т. е. F становится пустым файлом), устанавливает указатель файла на первую компоненту, присваивает функции EOF(F) значение TRUE. К функции REWRITE(F) необходимо обратиться *перед записью* в первую компоненту файла F.

11) GET(F) — в случае, когда EOF(F)=FALSE, продвигает указатель файла к следующей компоненте и присваивает значение этой компоненты «окну» F↑; если эта компонента — «конец файла», то значение F↑ становится неопределенным, а EOF(F) принимает значение TRUE.

12) PUT(F) — заносит значение F↑ в ту компоненту файла, куда установлен указатель, если значение EOF(F) есть TRUE к

*) Подробнее о файлах см. п. 21.

моменту выполнения PUT(F). После выполнения PUT(F) значение F↑ становится неопределенным.

Пример.

```
VAR DATA:FILE OF INTEGER;
  A:INTEGER;
BEGIN
  . . . . . .
  A:=SQR(DATA↑);
    GET (DATA);
    . . . . . .
```

Здесь оператор A:=SQR(DATA↑) присваивает переменной A квадрат текущей компоненты файла DATA; оператор GET (DATA) продвигает указатель к следующей компоненте файла и читает ее в переменную DATA↑.

13) WRITELN(F) — записывает символ «конец строки» в текущую компоненту текстового файла F.

14) READLN(F) — пропускает остаток текущей строки, устанавливает указатель файла на начало следующей строки текстового файла F.

15) EOLN(X) — принимает значение TRUE, если указатель файла установлен на символ конца строки (FALSE в противном случае) и засылает в X пробел.

16) EOF(X) — принимает значение TRUE, если X соответствует концу файла (FALSE — в противном случае).

17) NEW(P) — отводит место в памяти для новой динамической переменной и в P запоминает ее адрес (см. 22.1).

18) NEW(P,P1) — отводит место в памяти для записи (RECORD) с вариантом P1.

19) DISPOSE(P) — стирает динамическую переменную, на которую указывает переменная P.

20) DISPOSE(P,P1) — стирает динамическую переменную, созданную процедурой NEW(P,P1) и на которую указывает переменная P (см. 22.3).

16. О кодировке символов

1. Для ЭВМ БЭСМ-6 все символы, кроме ↑, соответствуют символам клавиатуры дисплея и перфоратора ЕС-9080 (кодировка КПК-12). Символ ↑ соответствует символу @ (коммерческое А). Если карты пробиты в кодировке КПК-12, то для БЭСМ-6 с операционной системой «Дубна» в качестве первой карты надо положить карту с пробивками 7/9 IBM (7/9 — в режиме MP пробить 7 и 9 в первой позиции).

2. На ЕС ЭВМ используется следующее представление символов:

Стандартное	[	]	{	}	AND	OR	NOT	<>
Паскаль ЕС	(.	.)	(*	*)	&	!	¬	¬=

17. Дополнительные сведения о языке паскаль для ЕС ЭВМ

17.1. Как читать листинг задачи

Каждая страница листинга начинается с информации о версии транслятора, дате и времени запуска программы.

Затем выдается информация о режиме трансляции. Например, если задан режим B+, то следующая строка будет INITIAL OPTIONS:B+.

Далее идет программа на паскале (см. пример).

Первая вертикальная колонка чисел слева — номера соответствующих строк программы.

Следующая вертикальная колонка четырехзначных шестнадцатеричных чисел содержит относительные адреса команд и переменных.

Для переменных в разделе VAR этот адрес указывает относительное смещение переменной от начала транслированной процедуры. В теле процедуры (и программы) адрес — есть относительный адрес от начала процедуры. Относительные адреса используются в ссылках раздела ERROR MESSAGE — списка диагностик транслятора.

Следующая вертикальная колонка состоит из двух символов — индикаторов вложений циклов и составных операторов. Когда в программе встречается первый оператор BEGIN, то вместо тире слева выдается ноль. Когда в какой-либо строке появляется END, соответствующий этому BEGIN, то в правом разряде двузначного «числа» появляется ноль.

Если внутри первой конструкции появляется вторая (BEGIN, CASE, REPEAT), то в левом разряде выдается 1, а для соответствующего END(UNTIL) в правом разряде появляется 1. Правильно составленная программа должна начинаться нулем в левом разряде индикатора вложения у первого оператора BEGIN программы и оканчиваться нулем в правом разряде у последнего оператора END. Если в программе есть раздел процедур и функций, то на листинг выводится буква — индикатор вложения модулей (процедур и функций). Эта буква выводится слева от заголовка модуля

и его BEGIN и END. Для уровня вложения 2 принята буква *A*, для уровня вложения 3 — буква *B* и т. д. Этот индикатор делает наглядной структуру процедур и позволяет сразу увидеть ошибку в случае пропуска END в конце модуля.

Пример.

```
1 0680  — —    PROGRAM L10T (INPUT,OUTPUT);
2 06A8  — —    VAR CH:CHAR;
3 0000   0 —   BEGIN
4 0062  — —      WHILE NOT EOF DO;
5 008A  1 —        BEGIN WRITE (' ');
6 0002  — —      WHILE NOT EOLN DO;
7 00EE  2  2       BEGIN   READ(CH);WRITE(CH) END;
8 016C  — 1      WRITELN;READLN END
9 0184  — 0    END.
```

17.2. Ошибки, обнаруживаемые при трансляции

Рассмотрим пример

```
1 0680  — —    PROGRAM L6T1(INPUT,OUTPUT),
               ***ERROR***                    !—14,18
2 06A8  — —    VAR A,B,R:REAL; I:INTEGER;
3 0000   0 —     BEGIN
. . .          . . . .
11 0196 — 0      END.
```

Ошибки, обнаруженные при трансляции, отмечаются в тексте программы символом «!», выдаваемым под ошибкой, и номером ошибки.

Ниже текста программы печатается расшифровка всех найденных ошибок (ERROR MESSAGE). Так, в приведенном примере будет напечатано:

14: Пропущена «;» (возможно, в предыдущей строке).
18: Ошибка в декларативной части.

17.3. Коды завершения трансляции

0 — трансляция закончена, ошибок нет.
4 — выдана предупредительная диагностика, фатальных ошибок нет, программа может быть выполнена.
8 — были фатальные ошибки, программа не выполняется.
12 — ошибка системы.
16 — ошибка транслятора.

Ошибки 12 и 16 могут быть следствием ошибок в управляющих картах (например, мало заказано памяти, не заказан соответствующий транслятор и т. д.).

Если управляющие карты правильные, то при кодах 12 и 16 необходимо обратиться к консультанту. Подробнее о кодах завершения см. [6].

17.4. Внутреннее представление данных

Тип	Число байт	Величина
INTEGER	4	32 бита.
BOOLEAN	4	1 = TRUE; 0 = FALSE.
CHAR	4	EBCDIC — код в младшем (правом) байте.
REAL	8	Число в форме с плавающей запятой.
SET	8	Элементы, представляемые побитно, начиная слева.
SCALAR	4	Порядковый номер величины в типе, начиная с нуля.

В упакованных структурах скалярные типы занимают чаще всего один байт, если порядковый номер упакованных элементов лежит в диапазоне от 0 до 255.

Пример.

```
X:INTEGER (занимает 4 байта).
X:PACKED ARRAY [1..4] OF CHAR (занимает 4 байта).
B:ARRAY [1..4] OF CHAR (занимает 16 байт).
C:REAL (занимает 8 байт).
```

18. Диагностика ошибок, обнаруженных при трансляции

В случае обнаружения ошибки в тексте программы, транслятор выдает соответствующее сообщение (диагностику).

На ЭВМ БЭСМ-6 подробная диагностика появляется прямо в тексте программы перед ошибочным оператором, ошибка отмечается символом «0».

На ЕС ЭВМ в тексте программы под ошибочным оператором появляется только номер допущенной ошибки и символ «!», отмечающий место ошибки в программе. Тексты подробных сообщений об ошибках выдаются единым массивом ниже текста всей программы.

После того как транслятор обнаружил ошибку, он пытается возобновить анализ программы, пропустив часть текста до ожидаемого символа. Часто удается успешно продолжить трансляцию; иногда это может привести к *наведенным* ошибкам.

Например, пусть при описании переменной I допущена следующая ошибка:

```
I,INTEGER;
```

т. е. вместо знака «:» стоит запятая.

Транслятор выдаст сообщение об ошибке, а затем будет «ругать» каждый оператор, использующий переменную I. Эти ошибки — наведенные, они являются следствием того, что тип переменной I описан в ошибочном операторе. Ошибки, допущенные в программе, бывают двух видов: *фатальные* и *нефатальные*. Если ошибка нефатальная, транслятор выдает предупредительную диагностику, но программа *выходит на счет*. Если допущена фатальная ошибка, программа *на счет не выходит*.

18.1. Сообщения об ошибках

Приведем список всех сообщений об ошибках, которые может обнаружить транслятор.

1: Ошибка в простом типе.
2: Пропущен идентификатор.
3: Пропущен оператор.
4: Пропущена ')'.
5: Пропущено ':'.
6: Недопустимый символ (возможно пропущен символ ';').
7: Ошибка в параметрах.
8: Пропущено OF.
9: Пропущена '('.
10: Ошибка в типе.
11: Пропущена '[' (или '(.').
12: Пропущена ']' (или '.)').
13: Пропущен END.
14: Пропущена ';' (возможно, строкой выше).
15: Должно быть INTEGER.
16: Пропущено '='.
17: Пропущен BEGIN.
18: Ошибка в разделе описаний.
19: Ошибка в списке полей.
20: Пропущена запятая.
21: Пропущена точка.
21*: Пропущена переменная *).
50: Ошибка в константе.
51: Пропущен знак ':='.
52: Пропущено THEN.

---

*) Номера ошибок со звездочкой приводятся для ЕС ЭВМ.

53: Пропущено UNTIL.
54: Пропущено DO.
55: Пропущено TO или DOWNTO.
56: Пропущено IF.
57: Пропущено слово FILE.
58: Ошибка в множителе (ошибочное выражение).
59: Ошибка в переменной.
60: Пропущено IN.
101: Дважды описанный идентификатор.
102: Нижняя граница больше верхней.
103: Идентификатор не принадлежит соответствующему классу.
104: Неописанный идентификатор.
105: Здесь знак не допускается.
106: Пропущено число.
107: Несовместимые ограниченные типы.
108: Здесь FILE не допускается.
109: Здесь тип не может быть REAL.
110: Тип переключателя должен быть скалярным или ограниченным.
111: Тип несовместим с типом переключателя.
112: Тип индекса не может быть REAL.
113: Индекс должен иметь скалярный тип либо ограниченный.
114: Базовым типом не может быть REAL.
115: Базовым типом должен быть скалярный либо ограниченный.
116: Ошибка в типе параметров стандартной процедуры.
117. Недопустимая ссылка на еще не описанное понятие.
118: Неописанный тип используется при описании переменной.
118*:Распаковку/упаковку применять нельзя; проверьте элементы массива.
119: Повторное описание списка параметров не допускается.
120: Тип результата функции может быть скалярным, ограниченным либо POINTER.
121: Параметр-значение не может быть файлом.
122: Функция уже декларирована (FORWARD); не допускается повторное описание типа результата функции.
123: Пропущен тип результата в описании функции.
124: Формат F допустим только для REAL.
125: Ошибка в типе параметра стандартной функции.
126: Число параметров иное, чем в описании функции (процедуры).
127: Недопустимые фактические параметры.
128: Тип результата параметра-функции не соответствует описанию.
129: Несовместимость типов операндов.
130: Тип выражения — не SET.
131: Допускается проверка только на равенство.
132: Не допускается строгое включение.
133: Не допускается сравнение файлов.

134: Недопустимый тип операнда.
135: Операнд должен иметь тип BOOLEAN.
136: Тип элемента множества скалярный или ограниченный.
137: Несовместимые типы элементов множества.
138: Тип переменной — не массив.
139: Тип индекса не соответствует описанию.
140: Тип переменной — не RECORD.
141: Тип переменной должен быть FILE либо POINTER.
142: Недопустимые типы фактических параметров.
143: Недопустимый тип переменной цикла.
144: Недопустимый тип выражения.
144*:Тип переключателя скалярный или ограниченный.
145: Несоответствие типов.
145*:Несовместимость с типом управляющей переменной.
146: Нельзя использовать файлы в операторе присваивания.
147: Тип метки не соответствует типу выражения переключателя.
148: Границы диапазона должны иметь скалярный тип.
149: Индекс может иметь тип ограниченный, но не INTEGER.
150: Нельзя присваивать значения стандартным функциям.
151: Нельзя присваивать значение формальному параметру-функции.
152: В данной записи нет такого поля.
153: Ошибка в типе параметра READ.
154: Фактический параметр должен быть переменной.
154*:Пропущена переменная.
155: Переменная цикла не может быть формальным параметром.
156: Одинаковые метки в CASE.
157: Слишком много вложенных операторов CASE.
158: Пропущено описание соответствующего варианта.
159: Типы REAL и строка недопустимы для переключателя.
160: Пропущено описание FORWARD.
161: Повторно описано FORWARD.
162: Размер памяти, занимаемый параметром, должен быть фиксированным.
162*:Внешние модули не могут быть описаны как FORWARD.
163: Пропущен вариант в описании.
164: Не допускается подстановка стандартной процедуры (функции).
165: Метки не должны повторяться.
166: Дважды описанная метка.
167: Неописанная метка.
168: Отсутствует метка, описанная в LABEL.
169: Ошибка в базовом типе SET.
170: Параметр должен быть параметром-значением.
170*:Тип переменной должен здесь быть ограниченным, скалярным либо POINTER.

171: Стандартный файл не требует описания.
172: Неописанный внешний файл.
173: Должна быть фортранная процедура или функция.
174: Должна быть паскаль-процедура или функция.
175: В заголовке PROGRAM пропущен файл INPUT.
176: В заголовке PROGRAM пропущен файл OUTPUT.
177: Здесь не допускается присваивание имени функции.
177*:Переменная цикла FOR должна быть локальной.
178: Дважды описанный вариант в RECORD.
178*:За именем файла данных не может следовать «/» в заголовке программы.
179*:Отсутствует оператор присваивания значения функции ее идентификатору.
180: Переменная цикла не может быть формальным параметром.
180*:Слишком длинная исходная строка.
181: Значение переключателя — вне диапазона.
182: Присваивание идентификатору функции должно стоять внутри FUNCTION.
183: Не происходит присваивания идентификатору функции в области действия этой функции.
185: Оператор присваивания идентификатору функции должен находиться в блоке этой функции.
186: Ошибка в заголовке процедуры; несоответствие фактических и формальных параметров по числу или по типу.
186*:Ошибка в заголовке процедуры.
187: Компоненты упакованных переменных не могут быть параметрами-переменными.
188: Идентификатор должен быть описан раньше, чем использован в других описаниях.
189: Ошибка в ширине поля для форматного ввода/вывода.
190: Нельзя изменять оператором присваивания переменную цикла.
191: Переменная цикла не может быть фактическим параметром-переменной.
201: Ошибка в вещественной константе (возможно, в ее записи присутствуют не только цифры, либо отсутствует точка).
202: Константа типа строка не должна выходить за пределы исходной строки.
203: Слишком большая целая константа.
204: Нулевая строка не допускается.
204*:В восьмеричном числе не может быть цифр 8 и 9.
205: Нулевая строка не допускается.
205*:Величина либо диапазон элементов SET ошибочны.
220: Инициализация переменных допускается только в главной программе.
221: Несоответствие типов при инициализации переменных.

222: Несоответствие числа компонент описанию структурной константы.
223: Несоответствие типов компонент описанию структурной константы.
224: Недопустимый формат структурной константы.
225: Отсутствует '*)' (т. е. правая фигурная скобка).
226: Недопустимый тип структурной константы.
227: Недопустима запись с вариантами для структурных констант.
250: Слишком много уровней вложения областей действия идентификаторов.
251: Слишком много вложенных процедур и (или) функций.
252: Слишком много FORWARD.
253: Слишком длинная процедура.
254: Слишком много констант в этой процедуре.
254*:Слишком большие переменные.
255: Слишком много ошибок в этой строке.
256: Слишком много внешних ссылок.
257: Слишком много EXTERNAL.
258: Слишком много локальных файлов.
259: Слишком сложное выражение.
260: Слишком много ENTRY.
261: Слишком много процедур либо переходов к глобальным меткам.
280: Имя EVENT не декларировано.
281: Нельзя использовать оператор POSTLUDE для события EXIT.
282: Дважды описанный оператор POSTLUDE.
292: Нельзя изменять тип константы.
293: Режим U устанавливает номер позиции, с которой начинается комментарий.
300: Деление на ноль.
300*:Инициализация величин не допускается во внешних модулях.
301: Нет поля CASE для данного значения.
302: Значение индекса вышло из диапазона.
303: Присваиваемое значение вышло из диапазона.
304: Значение элемента SET вышло из диапазона.
305: 'NIL' можно засылать только в переменную типа POINTER.
306: Тип POINTER не может указывать на переменную, содержащую поле типа FILE;
310: Внутри комментария символ ';' либо '(*'.
311: На метку, описанную в разделе LABEL, нет ссылки.
320: Структурная константа — расширение стандартного паскаля.
321: Раздел инициализации переменных — расширение стандартного паскаля.
322: Оператор FORALL — расширение стандартного паскаля.

323: Оператор LOOP — расширение стандартного паскаля.
324: Упрощенная форма оператора CASE — расширение стандартного паскаля.
325: Задание диапазона меток в операторе CASE — расширение стандартного паскаля.
326: Стандартный паскаль допускает здесь только идентификатор типа.
327: Использование оператора ** — расширение стандартного паскаля.
328: Функция изменения типа—расширение стандартного паскаля.
329: Спецификация интерактивного файла — расширение стандартного паскаля.
330: Ввод строки символов — расширение стандартного паскаля.
331: Автоматическая инициализация переключателя — расширение стандартного паскаля.
332: Расширение стандартного паскаля; в стандартном паскале этот тип надо описать.
333: Расширение стандартного паскаля; в стандартном паскале эту процедуру надо описать.
334: Расширение стандартного паскаля; эту функцию в стандартном паскале надо описать.
335: Процедура ORD(POINTER) — расширение стандартного паскаля.
336: 'EXTERN', 'PASCAL', 'FORTRAN' — расширение стандартного паскаля.
337: Стандартный паскаль требует описания внешнего файла в главной программе.
338: Метка должна содержать не более четырех цифр в стандартном паскале.
339: Стандартный паскаль не допускает использование символа $ (¤) в идентификаторе.
380: Нельзя передавать процедуру или функцию во внешний модуль.
381: Ошибка в типе результата внешней функции.
382: Нельзя отменять режим (* ¤E + *).
383: Можно транслировать только две процедуры одного уровня вложения после установления режима E.
384: Фактический параметр не может быть фортранным модулем.
385: Нельзя устанавливать режим E после трансляции внутренней процедуры.
389: Ограничение, накладываемое реализацией системы.
389*:Неожиданное появление оператора END.
399: Массивы переменной размерности не реализованы в системе.
399*:Не реализовано в системе.
400: Ошибка транслятора. Обратитесь к консультанту.

ГЛАВА II

ДЛЯ ТЕХ, КТО РЕШИЛСЯ ИДТИ ДАЛЬШЕ

19. Записи (RECORD)

Запись — это структура, состоящая из фиксированного числа компонент, называемых полями. В одном поле данные имеют один и тот же тип, а в разных полях могут иметь разные типы. Общий вид описания типа RECORD:

```
TYPE T=RECORD
  ID11,ID12,...ID1N:TYPE1;
  ID21,ID22,...ID2L:TYPE2;
  . . . . . .
  IDK1,IDK2,...IDKM:TYPEK
    END;
```

Здесь IDIJ — идентификаторы полей; TYPEI — типы полей; T — имя типа.

Пример 1. Данные комплексного вида можно описать переменной типа RECORD.

```
TYPE COMPLEX=RECORD
  RE,IM:REAL
  END;
VAR C:COMPLEX;
```

Здесь COMPLEX — имя типа, а C — имя переменной. Переменная C состоит из двух полей: RE и IM, имеющих один и тот же тип (REAL). Эти поля переменной C обозначаются как C.RE и C.IM.

Пример 2. Даты каких-либо событий можно описать следующим образом:

```
TYPE DATE=RECORD
  MONTH:1..12;
  DAY:1..31;
  YEAR:INTEGER
    END;
VAR D:DATE;
```

В этом примере описан тип DATE и переменная D, принадлежащая этому типу.

Переменная D описана как запись, состоящая из трех полей: MONTH, DAY и YEAR. Каждое поле содержит соответственно данные: целое число в пределах от 1 до 12 (номер месяца), целое число от 1 до 31 (число), целое число (год).

Поле DAY переменной D записывается как D.DAY.

Например, чтобы заслать в D дату 12.01.1985, надо выполнить следующие операторы:

```
D.MONTH:=1;
D.DAY:=12;
D.YEAR:=1985;
```

Пример 3. Вычислить сумму S двух комплексных чисел $X = 2 + 7i$ и $Y = 6 + 3i$ (т. е. X,Y,S:COMPLEX; см. пример 1). Фрагмент программы выглядит так:

```
X.RE:=2.0; X.IM:=7.0;
Y.RE:=6.0; Y.IM:=3.0;
S.RE:=X.RE + Y.RE;
S.IM:=X.IM + Y.IM;
```

Записи, как и массивы, могут быть упакованными.

Пример 4.

```
X:PACKED RECORD
  A:1..5;
  B:CHAR
    END;
```

Запись может быть компонентой других структур. Например, введем тип FAMILY (семья: отец, мать, 1-й ребенок, 2-й ребенок):

```
TYPE FAMILY=(FATHER,MOTHER,
CHILD1,CHILD2);
VAR BIRTHDAY:ARRAY [FAMILY] OF DATE;
```

где DATE — описанная выше запись.

Переменная BIRTHDAY есть массив, состоящий из записей — дат рождения членов семьи: отца, матери, 1-го ребенка, 2-го ребенка. Каждая дата рождения имеет тип DATE, который должен быть описан в программе.

Для занесения даты рождения, например, матери (MOTHER), достаточно выполнить операторы:

```
BIRTHDAY[MOTHER].MONTH:=5;
BIRTHDAY[MOTHER].DAY:=1;
BIRTHDAY[MOTHER].YEAR:=1950;
```

Занесется дата рождения матери — 1.5.1950.

19.1. Оператор WITH

Этот оператор используется для удобства работы с переменными типа RECORD (запись).

Общий вид:

```
WITH A DO ST;
```

здесь A — имя переменной типа RECORD, ST — оператор.

В операторе ST при ссылках на компоненты записи имя A можно опускать.

Пример. Для занесения даты рождения в предыдущем примере достаточно выполнить операторы:

```
WITH BIRTHDAY[MOTHER] DO
BEGIN
      MONTH:=5;
      DAY:=1;
      YEAR:=1950
END;
```

19.2. Запись с вариантами

Общий вид:

```
TYPE V=RECORD
  A:TYPE1
  B:TYPE2
  . . . . .
  CASE N:TYPEN OF
    C1:(T11:TYPE11;
      T12:TYPE12;...);
    C2:(T21:TYPE21;
      T22:TYPE22,...);
      . . . . . .
    CK:(TK1:TYPEK1;
      TK2:TYPEK2;...)
END;
VAR Z:V;
```

Здесь Z — переменная типа V; N — переменная, называемая *переключателем*; TYPEN — тип переменной N.

Этому же типу должны принадлежать метки C1,C2,...CK. Каждой метке соответствует набор *полей* T11, T12,... Эти поля являются *компонентами варианта*.

Переменную N называют также тагом (тэгом), ярлыком, признаком, дискриминантом.

Если какой-либо метке CL вообще не соответствуют поля, то пишут CL:( );

Замечания.

1. Любая запись (RECORD) может иметь только одну вариантную часть (CASE).

2. Вариантная часть должна помещаться после постоянной части.

3. Среди идентификаторов полей не должно быть одинаковых. Обращение к компоненте Z.Tij записи Z происходит так:

1) Присваивается соответствующее аначение (C_i) переключателю N. В зависимости от значения N переменная Z, помимо полей A,B,..., содержит те поля, которые соответствуют той метке C_i, с какой совпадает аначение N.

2) Выполняется операция с компонентой Z.Tij.

Пример записи с вариантами. Пусть необходимо собрать следующие сведения о сотрудниках института: фамилию, дату рождения и, если есть семья, то фамилию и дату рождения супруги (супруга).

Эта информация может быть описана, например, записью PERSON.

Пусть переменная типа KIND может иметь одно иа значений («женат», «холост»).

```
KIND=(MARRIED,SINGLE);
  PERSON=RECORD
     NAME:ALFA;
     DATEBIRTH:DATE;
     CASE YESNO:KIND OF
       MARRIED:(NAME1:ALFA;DATE1:DATE);
       SINGLE:( )
     END;
```

Здесь NAME — строка символов (например, 'ROGOV_'); DATEBIRTH — запись, описанная выше, содержит дату рождения (например, 15.02.62); YESNO — переключатель типа KIND, который может принимать одно из двух аначений: MARRIED либо SINGLE; NAME1 — строка символов, содержащая фамилию супруги (супруга) (например, 'ROGOVA'); DATE1 — запись, содержащая дату рождения супруги (супруга). SINGLE — пустое поле.

Если ROGOV женат, то присутствует поле MARRIED, если холост — поле SINGLE, а поле MARRIED отсутствует. Паскаль допускает вложение вариантов в типе RECORD.

Пример. Пусть необходимо, помимо информации предыдущего примера, иметь о сотрудниках следующие сведения: если сотрудник холост, но состоял в браке раньше, то укаааать, когда разведен.

Опишем тип KIND как (женат, холост, разведен, нет):

```
KIND=(MARRIED,SINGLE,DEVORCED,NO);
```

Тогда

```
    PERSON=RECORD
      NAME:ALFA;
      DATEBIRTH:DATE;
      CASE YESNO:KIND OF
        MARRIED:(NAME1:ALFA; DATE1:DATE);
        SINGLE: (CASE YN:KIND OF
          DEVORCED:(DATEDV:DATE));
          NO:( )
      END;
```

Здесь для варианта SINGLE имеется вложенная запись с вариантами DEVORCED (разведен) и NO.

Если сотрудник состоит в браке, то в записях информации отсутствует поле SINGLE; если разведен, то отсутствует MARRIED; если в браке не состоял, то запись содержит лишь поле NAME, DATEBIRTH и пустое поле NO.

З а м е ч а н и е. Перед засылкой информации в запись программист должен присвоить переключателю соответствующее значение. В противном случае информация (например, MARRIED) в поле заслана не будет, и система никакой диагностики не выдаст.

Пример засылки информации о сотруднике РОГОВЕ, родившемся 1.12.32, женатом на РОГОВОЙ, родившейся 15.3.30 (ЭВМ БЭСМ-6).

```
PROGRAM L8T3(INPUT,OUTPUT);
TYPE KIND=(MARRIED,SINGLE);
    DATE=RECORD
      DAY:1..31;
      MONTH:1..12;
      YEAR:INTEGER
      END;
    PERSON=RECORD
      NAME:ALFA;
      DATEBIRTH:DATE;
      CASE YESNO:KIND OF
        MARRIED:(NAME1:ALFA; DATE1:DATE);
        SINGLE:(NAMESING:ALFA)
      END;
VAR P:PERSON;
BEGIN
    WITH P DO
      BEGIN
        YESNO:=MARRIED;
        NAME:='ROGOV ';
```

```
      WITH DATEBIRTH DO
        BEGIN
          DAY:=1;
          MONTH:=12;
          YEAR=1932
        END;
      CASE YESNO OF
        MARRIED:BEGIN
              NAME1:='ROGOVA';
              WITH DATE1 DO
               BEGIN
                 DAY:=15;
                 MONTH:=3;
                 YEAR:=1930
               END
             END;(* MARRIED *)
        SINGLE:NAMESING:='SINGLE';
      END (* CASE YESNO OF *)
    END (* WITH P DO *)
  WITH P DO
    WRITE ('␣', NAME);
  WITH P.DATEBIRTH DO
    WRITELN ('␣',DAY,'/',MONTH,'/',YEAR);
  WITH P DO
    WRITE ('␣',NAME1);
  WITH P.DATE1 DO
    WRITELN ('␣',DAY,'/',MONTH,'/',YEAR)
END.
```

Более сложная программа, иллюстрирующая работу с переменными типа «запись», приведена в Приложении 1.

20. Множества (SET)

Рассмотрим несколько примеров.

Пример 1. Пусть в нашем распоряжении имеется множество из трех монет разного достоинства: 1 к, 5 к, 10 к. Из этих монет можно составить следующие подмножества (их число равно $2^3 = 8$):

1) {1 к.};
2) {5 к.};
3) {10 к.};
4) {1 к, 5 к.};
5) {1 к, 10 к.};
6) {5 к, 10 к.};
7) {1 к, 5 к, 10 к.};
8) пустое подмножество ϕ.

Эти подмножества и будут принадлежать некоторому множеству, тип которого назовем SUM; сами элементы (монеты), из которых со-

ставляется подмножество, пусть принадлежат некоторому базовому типу, который назовем MONET.

Опишем типы данных этого примера:

```
TYPE MONET=(KOP1,KOP5,KOP10);
  SUM=SET OF MONET;
```

Пример 2. Рассмотрим в качестве элементов базового типа сигналы от 4-х абонентов (AB1, AB2, AB3, AB4), поступающие на телефонную станцию. Обозначим базовый тип через ABONENT:

```
TYPE ABONENT=(AB1,AB2,AB3,AB4),
```

тогда комбинации сигналов можно описать переменной типа «множество». Назовем этот тип SIGN:

```
SIGN=SET OF ABONENT;
```

Тип SIGN описывает такие комбинации:

1) только AB1; 2) только AB2; 3) только AB3; 4) только AB4 5) AB1 и AB2; 6) AB1 и AB3; 7) AB1 и AB4; 8) AB2 и AB3; 9) AB2 и AB4; 10) AB3 и AB4; 11) AB1, AB2 и AB3; 12) AB1, AB2 и AB4 13) AB1, AB3 и AB4; 14) AB2, AB3 и AB4; 15) AB1, AB2, AB3 и AB4; 16) отсутствие сигналов;

В общем виде тип «множество» описывается так:

```
TYPE A=SET OF TC;
```

Здесь A — идентификатор типа (произвольный); TC — тип компонент множества, называемый базовым типом.

Значение переменной типа «множество» изображается путем перечисления конкретных компонент, разделенных запятыми и заключенных в квадратные скобки.

Пример 3. Пусть базовый тип INT и тип A заданы так:

```
TYPE INT=1..3;
  A=SET OF INT;
```

Переменная типа A в этом случае может принимать следующие значения: [1], [3], [2], [1, 3], [1, 2], [3, 2], [1, 3, 2], [ ], где [ ] означает пустое множество. Например, если переменная B имеет тип A, то можно присвоить ей одно из значений: B:=[1,3]; B:= [1,3,2]; и т. д.

Пример 4. Если базовый тип описывает набор двоичных бит, то можно получить их комбинацию. Пусть

```
TYPE BIN=(BIT1,BIT2,BIT3);
     BTS=SET OF BIN;
```

Переменная типа BTS может принимать значения: [BIT1],[BIT2], [BIT3],[BIT1,BIT2],[BIT1,BIT3],[BIT2,BIT3],[BIT1,BIT2,BIT3], [ ].

Таким образом, используя переменные типа SET, можно работать с битовой информацией.

Паскаль допускает множества, состоящие из ограниченного числа элементов $N \leqslant NMAX$, где NMAX — машинозависимая константа:

на БЭСМ-6 NMAX = 48,
на ЕС ЭВМ NMAX = 63.

В качестве базового типа можно использовать любой простой тип, кроме REAL. Если в качестве базового типа выбран тип CHAR, то допускается использовать не более NMAX первых символов, имеющихся в распоряжении транслятора. Если задача требует использования множества, состоящего из большего числа элементов, то его можно представить как массив множеств, состоящих из допустимого числа элементов.

Для ограниченных типов от базового типа INTEGER можно использовать в качестве компонент целые числа от нуля до $N \leqslant \leqslant NMAX - 1$

Пример 5.

```
TYPE N=(0..62);
VAR P:SET OF N;
```

20.1. Данные типа SET

Данные типа SET задаются путем перечисления значений, разделенных запятыми и заключенных в квадратные скобки.

Общий вид:

[EXPR1,EXPR2,...EXPRN];

Здесь EXPRI — выражения базового типа.
Порядок следования выражений — несущественен. Непустой набор может быть также выражением вида:

[EXPR1]
[EXPR1..EXPRK]
[EXPR1,EXPRK..EXPRN]

Данные вида [EXPR1..EXPRK] соответствуют набору всех элементов базового типа от значения EXPR1 до EXPRK.

Пример 1. [3*6—7..15+4] соответствует набору [11..19], т. е.

[11,12,13,14,15,16,17,18,19].

Если окажется, что для [I..J], $I > J$, то такое множество интерпретируется как пустое, а в случае $I = J$ — как множество, содержащее один элемент — I.

Пример 2. [3*6—7..5+6] эквивалентно [11].

Пример:

```
TYPE COLOR=(RED,YELLOW,GREEN,BLUE);
VAR MIX:SET OF COLOR;
. . . . . .
MIX:= [RED,BLUE];
```

Напомним, что на ЕС ЭВМ символам '[' и ']' соответствуют пары символов '(.' и '.)'.

Пример 4.

```
TYPE N=(1,3,5,7,9);
VAR K:SET OF N;
. . . . . .
  K:=[3..9];
```

В этом случае в K заплется комбинация [3,5,7,9];

20.2. Операции с переменными типа SET

К переменным типа SET применимы следующие операции:
=, < >, >=, <=, IN, +, −, *.

Операции = и < > используются для проверки эквивалентности: два значения переменной типа SET считаются равными, если они состоят из одних и тех же элементов.

Пример 1.

```
[1,3]          =  [3,1]          дает TRUE,
[1..3]         =  [1,2,3]        дает TRUE,
[1]            <> [2]            дает TRUE,
[1,2,3]        =  [1,4,3]        дает FALSE,
[RED, BLUE]    =  [RED, YELLOW]  дает FALSE.
```

Операции >= и <= используются для проверки принадлежности одного множества другому: так, если множество A содержится во множестве B, то A <= B дает TRUE.

Пример 2.

```
[1, 2] <= [1,2,3] дает TRUE
```

Пустое множество [ ] содержится во всех множествах, т. е. всегда [ ] <= [B] дает TRUE.

Операция IN используется для установления наличия определенного элемента в величине типа SET. Так, если X есть элемент множества B, то (X IN B) дает TRUE. Общий вид:

```
X IN A;
```

здесь X — величина базового типа, A — величина типа SET.

Пример 3.

RED IN [RED,YELLOW] дает TRUE;
RED IN [BLUE,GREEN] дает FALSE;

Замечание 1. Чтобы проверить, является ли значение N цифрой, удобно использовать операцию IN следующим образом:

IF N IN [0..9] THEN...

Замечание 2. Результат операции IN может быть неопределенным в некоторых случаях

Пример 4. Пусть

A: SET OF 1..50;
X: INTEGER.

Если заслать в X число, большее максимального значения 50 (например, X:= 55), то в этом случае результат операции X IN A не всегда FALSE.

К переменным типа SET, относящимся к одному и тому же конкретному типу, применимы операции:

\+ объединение;
* пересечение;
— дополнение.

Пусть A и B — операнды, имеющие один и тот же конкретный тип. Тогда

A + B представляет собой объединение множеств элементов, входящих в A и B (одинаковые элементы не повторяются).

A * B — пересечение множеств элементов A и B.

A — B — множество элементов, которые есть в A, но отсутствуют в B.

Пример 5.

[1,3] + [1,4] дает [1,3,4];
[1,3] * [1,4] дает [1];
[1,3] — [1,4] дает [3].

Операция A:= A + X добавляет элемент X к множеству A. Если X уже имелся в A, то множество A не меняется. A:= A — X исключает X из A. Если X отсутствовал в A, то множество A не меняется.

21. Файлы (FILE)

Для того чтобы понятие файла было для читателя достаточно ясным, рассмотрим процесс записи на магнитную ленту. Запись производится посредством магнитных головок на тот участок ленты, который находится против головки. Для простоты будем считать, что

головка может перемещаться вдоль магнитной ленты (в действительности головки закреплены, а лента движется).

Пример. Пусть записывающая головка (изобразим ее стрелкой) установлена против участка A магнитной ленты (рис. 2):

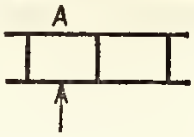


Рис. 2

Если на ленту будет записано некоторое число 153, то головка после записи переместится к следующему участку ленты (B) (рис. 3):

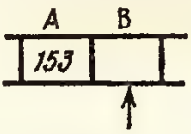


Рис. 3

Пусть на ленту таким образом будет записана последовательность чисел: 153, 512, 25, —13 (рис. 4)

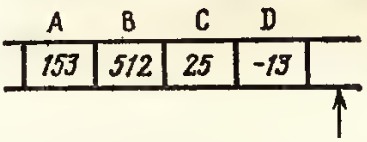


Рис. 4

В этом случае говорят, что на магнитной ленте находится файл (переменная типа FILE) длиной 4, а записывающая головка продвинулась за пределы файла.

Файл представляет собой последовательность компонент одного и того же типа. Число компонент не фиксировано. В каждый момент доступна только одна компонента. Говорят, что на эту компоненту установлен *указатель файла*.

Если выполнилась операция записи в n-ю компоненту файла, то указатель автоматически продвигается к $(n + 1)$-й компоненте, т. е. для записи становится доступной уже только $(n + 1)$-я компонента.

Длиной файла называется число записанных компонент. Файл, не содержащий компонент, называется *пустым*, его длина равна нулю. Читать файл можно также только последовательно по одной компоненте. Файлы паскаля поэтому называют *файлами последовательного доступа*. Начать писать в файл можно только *с самого его начала*, дописывая новые компоненты последовательно одну за другой; для чтения также надо начинать просмотр файла *с самого начала*.

Вследствие такой организации на одном просмотре файла нельзя совмещать и чтение, и запись информации: можно либо только читать, либо только писать.

Для того чтобы определять готовность файла к чтению либо к записи информации, существует стандартная функция EOF(F), где F — имя файла. Если указатель файла продвинулся за конец файла (готовность к записи), то эта функция принимает значение TRUE, во всех остальных случаях эта функция принимает значение FALSE.

Общий вид описания типа FILE:

```
TYPE R = FILE OF TC;
```

здесь R — идентификатор типа, TC — тип компонент (может быть любым, кроме типа FILE).

Файлы могут быть разных типов: состоять из целых компонент, либо вещественных, либо записей и т. д. Как и другие переменные, каждую переменную-файл надо описать в разделе VAR. Вводя имя переменной файла (имя файла), надо указать, какого типа файл. Этот тип должен быть обозначен каким-либо именем и описан в разделе TYPE.

Например, файл F вещественных чисел:

```
TYPE N=FILE OF REAL;
(* описание переменной файла *)
VAR F:N;
```

Файл может быть описан и непосредственно при описании переменной, например

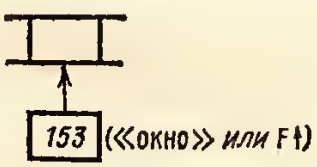
```
VAR F:FILE OF REAL;
```

В первом случае введено имя файла F и соответствующий тип обозначен N; во втором введено имя файла F, а его тип имени не имеет и поэтому в разделе TYPE не описывается.

Если переменная F имеет тип FILE, то транслятор автоматически вводит переменную F↑, называемую буферной переменной, «окном», через которое можно прочитать или записать одну компоненту. Переменная F↑ имеет тот же тип, что и компоненты файла F. «Окно» устанавливается против той компоненты, где находится указатель файла.

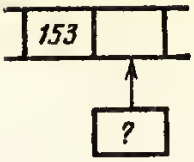
Пример. Рассмотрим на схеме запись в файл чисел 153, 512, 25.

1)



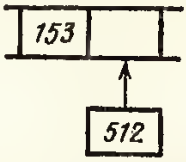
В «окно» F↑ записано число 153.

2)



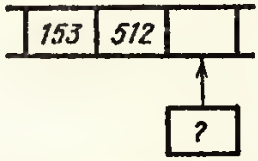
Выполнен приказ на запись в файл. Значение F| «испорчено», там уже ве 153.

3)



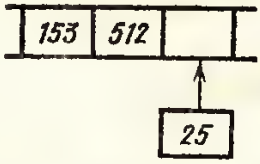
В «окно» занесено число 512.

4)



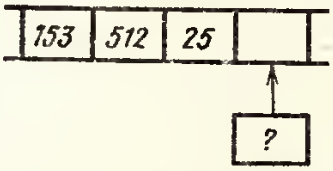
Выполнен приказ на запись в файл.

5)



В окно занесено число 25.

6)



Выполнен приказ на запись в файл.

После записи в файл значение буферной переменной F↑ становится неопределенным («портится»).

Файлы создаются не только на магнитных лентах, но и на магнитных дисках, других внешних устройствах, а также в оперативной памяти. Способы работы с такими файлами с точки зрения программы во всех этих случаях одинаковы: доступна только одна компонента файла (посредством «окна»). Файлы прямого доступа в стандарте языка паскаль отсутствуют [4].

Работа с файлами (F) проводится посредством следующих стандартных процедур.

1. RESET(F) — подготовка к чтению информации из файла с именем F. Эта процедура осуществляет следующие действия: если файл не пустой, то устанавливается указатель файла на 1-ю компоненту и эта компонента считывается в «окно» (F↑); функции EOF(F)

присваивается значение FALSE. Если файл был пустым, то значение F↑ «портится», а EOF(F) присваивается значение TRUE.

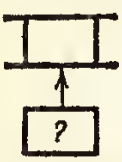
2. GET(F) — чтение компоненты файла F. Процедура выполняется только при условии, если EOF(F) имеет значение FALSE. Проверку этого условия необходимо сделать в программе перед обращением к GET(F). Процедура продвигает указатель к следующей компоненте файла и считывает эту компоненту в «окно» (F↑); если указатель выйдет за пределы файла, то функция EOF(F) принимает значение TRUE, а значение F↑ «испортится».

3. REWRITE(F) — подготовка к записи информации в начало файла F. Процедура очищает файл F и устанавливает указатель файла на 1-ю компоненту; функции EOF(F) присваивается значение TRUE.

4. PUT(F) — запись компоненты в файл F. Перед обращением к этой процедуре в программе необходимо проверить значение функции EOF(F): файл готов к записи только при значении TRUE. Процедура записывает в файл значение буферной переменной (F↑) и передвигает указатель за пределы файла, готовясь записать следующую компоненту. Значение EOF(F) остается TRUE, а F↑ — «портится».

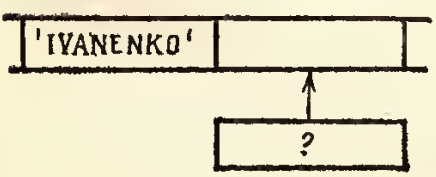
П р и м е р. Проиллюстрируем на примере работу процедур REWRITE и PUT для следующей задачи: записать в начало файла GRUPP2 следующие компоненты: 'IVANENKO','SHELKOV_'. Очевидно, файл содержит компоненты типа ALFA (см. 6.5.1), буферная переменная также типа ALFA. (Файл GRUPP2 содержал ранее другие значения).

1) Выполним процедуру REWRITE (GRUPP2):

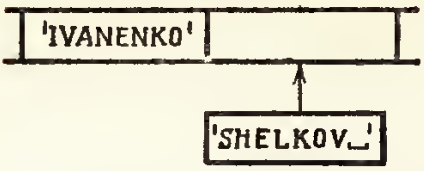


2) Занесем 1-е значение в буферную переменную
GRUPP 2↑ := 'IVANENKO';

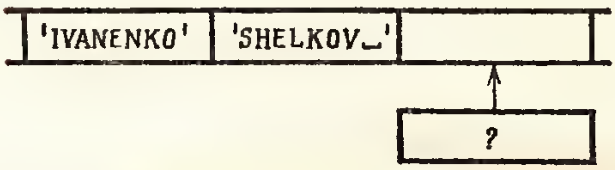
3) Выполним процедуру PUT (GRUPP2):



4) Занесем 2-е значение в буферную переменную:
GRUPP 2↑:='SHELKOV';



5) Выполним процедуру PUT(GRUPP2);



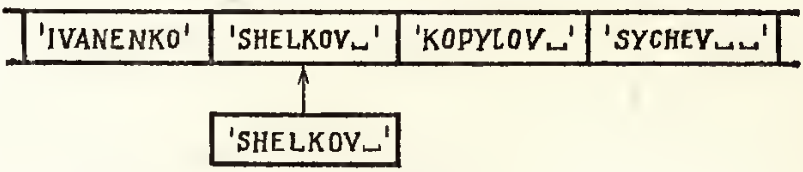
Для упрощения работы с файлами введены процедуры чтения информации из файла — READ и записи в файл — WRITE, которые освобождают программиста от манипуляций с буферной переменной.

Общий вид:

READ (F, X1, X2, . . ., XN);

здесь F — имя файла, X1, X2, . . ., XN — переменные, куда считываются компоненты файла, начиная с той компоненты, какая была занесена в «окно» файла.

Пример. Пусть состояние файла GRUPP2 соответствует схеме



Тогда после выполнения оператора

READ (GRUPP2,X1,X2,X3),

(где X1,X2,X3 должны иметь тип ALFA) картинка будет выглядеть так:

Значения переменных X1,X2,X3 будут следующими:

X1	X2	X3
'SHELKOV␣'	'KOPYLOV␣'	'SYCHEV␣␣'

Процедура записи в файл имеет вид

WRITE (F, A_1, A_2, . . . A_n)

Здесь F имя файла, A_1, A_2, . . . A_n — *выражения* того же типа, что и компоненты файла. Процедура записывает значения выражений A_1, . . ., A_K по одному в файл F, начиная с того места, куда был установлен указатель файла в момент обращения к процедуре WRITE.

Пример. Пусть A1='TEVELEV␣'; A2='RODIONOV', A3='MALYSHEV', а состояние файла GRUPP2 такое, каким оно было после выполнения оператора READ. Тогда после работы оператора WRITE(GRUPP2,A1,A2,A3) файл GRUPP2 будет таким:

'IVANENKO'	'SHELKOV␣'	'KOPYLOV␣'	'SYCHEV␣␣'	'TEVELEV␣'	'RODIONOV'	'MALYSHEV'

В стандарте языка паскаль работа с файлами через READ и WRITE допускается только для компонент, имеющих тип CHAR. Но трансляторы на ЕС ЭВМ и БЭСМ-6 не накладывают ограничений: для любых файлов можно использовать эти процедуры.

В качестве примеров рассмотрим три задачи, имеющие практическое значение.

Пример 1. Запись информации в файл F1 (переменная X должна иметь тип компонент файла). Фрагмент программы такой:

```
REWRITE(F1);
REPEAT
(* ПОЛУЧЕНИЕ ЗНАЧЕНИЯ X — ОЧЕРЕДНОЙ
   КОМПОНЕНТЫ ФАЙЛА *)
WRITE(F1,X)
UNTIL <условие окончания>
```

Пример 2. Чтение всей информации из файла F2 (переменная Y должна иметь тип компонент файла F2).

```
RESET(F2)
WHILE NOT EOF(F2) DO
BEGIN
  READ(F2,Y);
  (* ОБРАБОТКА ОЧЕРЕДНОЙ КОМПОНЕНТЫ *)
END;
```

Пример 3. Записать в файл F1 числа 2, 3, 4, 5, а в файл F2 числа 2!, 3!, 4!, 5! Затем прочитать из файлов F1 и F2 эти числа и напечатать.

```
//TEST5 JOB XXXXX,SEMASHKO,MSGLEVEL=(2,0)
// EXEC PASCLG
//PASC.SYSIN DD *
PROGRAM FL(INPUT, OUTPUT);
```

```
VAR I,J,X,Y:INTEGER;
    F1,F2:FILE OF INTEGER;
BEGIN
(* ЗАПИСЬ В ФАЙЛ *)
    REWRITE(F1); REWRITE(F2);
    X:=1; Y:=1;
    REPEAT
      X:=X+1; Y:=Y*X;
      WRITE(F2,Y); WRITE(F1,X)
    UNTIL X=5;
    (* ЧТЕНИЕ ИЗ ФАЙЛА *)
    RESET(F1); RESET(F2);
    WHILE NOT EOF(F1) DO
      BEGIN
        READ (F2, Y);
        READ (F1, X);
        WRITELN(' X= ,X, ' Y=',Y)
      END
END.
//GO.P@ LOCAL DD UNIT=SYSDA,
//                DISP=OLD,VOL=SER=RESMVT
//GO.SYSIN DD *
//
```

Результат работы программы:

X = 2; Y = 2;
X = 3; Y = 6;
X = 4; Y = 24;
X = 5; Y = 120.

Пример 4. Копирование файла F2 в файл F1. Компоненты файлов F1 и F2, а также переменная X должны иметь один и тот же тип.

```
RESET(F2);
REWRITE(F1);
WHILE NOT EOF(F2) DO
BEGIN
  READ(F2,X); WRITE(F1,X)
END;
```

21.1. Внешние файлы

Файлы по отношению к программе могут быть внешними и внутренними.

Внутренними файлами являются такие, которые создаются, используются и существуют только во время работы данной программы.

Однако часто возникает необходимость использовать файлы, созданные ранее и хранящиеся на внешних запоминающих устройствах. Например, программы, обрабатывающие данные физических экспериментов, вводят эти данные с файлов на магнитных лентах либо дисках; эти файлы были записаны ранее, возможно, в ходе самого эксперимента. С другой стороны, обработка данных физического эксперимента, как правило, представляет собой сложную задачу, решение которой ведется в несколько этапов. На каждом этапе получается файл промежуточных результатов, который используется впоследствии другими программами. Такие файлы, которые существуют вне программы, называют *внешними файлами*.

Внешние файлы указываются в списке параметров PROGRAM и соответственно описываются в управляющих картах (см. [6]).

21.2. Текстовые файлы

Особое значение имеют файлы, компонентами которых являются символы. Такие файлы называются *текстовыми*.

Тип текстового файла (TEXT) в каждом трансляторе с паскаля заранее предопределен как

```
TEXT=FILE OF CHAR
```

(описывать в программе этот тип не требуется). Текстовый файл состоит из последовательности строк, каждая из которых содержит величины типа CHAR и заканчивается специальным символом «конец строки» (EOL).

Стандартные файлы INPUT и OUTPUT являются текстовыми.

С символом «конец строки» оперируют следующие процедуры:

WRITELN(F) — записывается символ «конец строки» (EOL) в компоненту файла, на которую установлен указатель файла.

READLN(F) — пропускается оставшаяся часть текущей строки и указатель файла устанавливается на первый символ новой строки. В «окно» файла F↑ считывается этот символ.

EOLN(F) — функция принимает значение TRUE, если указатель установлен на символ «конец строки», и засылает пробел в F↑.

Если F — текстовый файл, а CH — символьная переменная, то можно использовать следующие формы процедур записи и чтения.

Для записи в текстовый файл F значения символьной переменной CH можно воспользоваться процедурой

```
WRITE (F,CH) вместо F↑:=CH; PUT(F);
```

Если V1,V2,V3, . . . VN — символьные переменные, то можно их значения записать в файл F процедурой

```
WRITELN(F,V1,V2,V3,...VN) вместо
WRITE(F,V1);WRITE(F,V2);...WRITE(F,VN);
WRITELN(F);
```

При этом следом за значением VN в файл F запишется символ «конец строки».

Аналогично можно воспользоваться оператором

```
READ (F, CH) вместо CH:=F↑; GET(F);
```

для чтения из текстового файла F.

Если надо прочитать N символов и перейти к новой строке файла F, то можно использовать оператор

```
READLN(F,V1,V2,V3,...VN) вместо
READ(F,V1); READ(F,V2);...READ(F,VN);
READLN(F);
```

21.3. Стандартные текстовые файлы INPUT и OUTPUT

Стандартный ввод и вывод в языке паскаль осуществляется с помощью текстовых файлов INPUT и OUTPUT (файлов стандартного типа TEXT), описанных в качестве параметров PROGRAM.

Для упрощения работы с такими файлами предоставлены дополнительные возможности: по умолчанию для переменной CH типа CHAR

```
WRITE(CH)  соответствует WRITE (OUTPUT, CH);
READ(CH)   соответствует READ (INPUT, CH);
WRITELN    соответствует WRITELN (OUTPUT);
EOF        соответствует EOF (INPUT);
EOLN       соответствует EOLN (INPUT);
READLN     соответствует READLN (INPUT).
```

К файлам INPUT и OUTPUT нельзя применять RESET и REWRITE.

Первый символ каждой строки файла OUTPUT управляет устройством печати и на печать не выводится. Если этот символ «пробел», то — переход к следующей строке; «0» — пропуск строки; «1» — переход к началу следующей страницы листинга; «+» — печать без перехода к новой строке (печать с наложением строк).

Пример 4. Печать значения A с повой страницы:

```
WRITE('1',A)
```

Пример 5. Печать содержимого внешнего текстового файла X.

```
PROGRAM L10T2 (OUTPUT,X);
VAR CH:CHAR;
```

```
  X:FILE OF CHAR;
BEGIN WRITE(' ');.
  RESET (X); WHILE NOT EOF(X) DO
  BEGIN
    WHILE NOT EOLN(X) DO
    BEGIN READ(X,CH); WRITE(CH) END;
    WRITELN; READLN(X)
  END
END.
```

Кроме того, для файлов INPUT и OUTPUT процедуры READ и WRITE позволяют работать с параметрами не только типа CHAR, но и параметрами типа BOOLEAN, REAL и INTEGER. Если первый параметр процедур READ и WRITE — текстовый файл, то информация из него читается или в него записывается; если первый параметр— не текстовый файл, то автоматически информация читается из файла INPUT или записывается в файл OUTPUT соответственно. Трансляторы ЕС ЭВМ и БЭСМ-6 позволяют также вводить и выводить переменные типа ALFA, строки символов.

22. Ссылки (POINTER)

До сих пор мы имели дело с такими переменными, которые описывались в разделе VAR какого-либо блока программ. Транслятор после анализа этого раздела отводит каждой переменной соответствующее число ячеек памяти и закрепляет их за данной переменной на все время работы блока. Такие переменные называют *статическими*. Они не могут быть использованы системой под другие нужды, даже если в процессе дальнейшей работы программы эти переменные больше не понадобятся.

Данные могут быть организованы и другим образом. Их можно хранить в некоторой области памяти, не обозначая именем переменной, а используя ссылку на эту область. Аналогично тому, как иногда «обозначают» зрителя: «человек с 5-го ряда, 3-го места». Как мы увидим ниже, такой вид доступа позволяет динамически захватывать и освобождать память в процессе работы блока программы. Поэтому и сами переменные, которые могут создаваться и ликвидироваться по мере надобности, называют *динамическими*.

Остановимся подробнее на работе с динамическими переменными, которые чаще всего реализуются как *связанные структуры*.

П р и м е р. Проиллюстрируем особенности такой связанной структуры на примере очереди на прием к врачу. Каждый пациент запоминает человека, за которым занял очередь. Все пациенты связаны в цепочку согласно очереди, но в пространстве они размещены произвольным образом: вновь подошедший садится на любое сво-

бодное место, т. е. соседние элементы очереди могут находиться на произвольном расстоянии в пространстве.

Подобным образом строится структура связанных данных, которые могут занимать память не подряд, а размещаться там, где есть свободное место. Каждый элемент такой структуры должен «знать», за кем он «стоит», т. е. содержать ссылку на предыдущий элемент цепочки.

Чтобы проиллюстрировать преимущества динамических переменных, продолжим аналогию с очередью пациентов.

Пусть один из пациентов покидает очередь. Этот процесс не требует перемещения в пространстве остальных пациентов: просто стоящий за ушедшим теперь запоминает другого человека. То есть исключение элемента из цепочки данных сводится к изменению одной единственной ссылки и не требует перемещения остальных элементов, как это имело бы место для массива. На рис. 5 показывается исключение элемента цепочки.

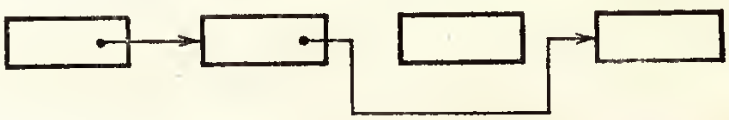


Рис. 5

Исключенный элемент теперь можно освободить от участия в работе блока и использовать занимаемый им участок памяти для других целей.

С помощью ссылок легко вставить новую компоненту в цепочку данных. Для этого достаточно изменить две ссылки (см. рис. 6).

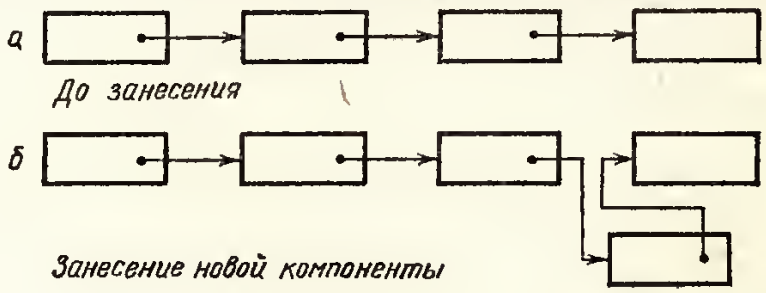


Рис. 6

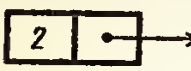
Новая динамическая компонента может быть размещена в любом свободном месте памяти, отведенном под такие переменные. Сама динамическая переменная не обозначается идентификатором. Динамическая переменная — это «невидимка» в программе: идентификатором она не обозначается, транслятор ей место в памяти не отводит. Память под такую переменную резервируется и освобождается динамически в процессе счета (с помощью специальных процедур).

Обращение к динамической переменной происходит посредством *ссылочной переменной*, которая содержит *адрес* соответствующей динамической переменной.

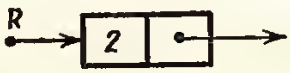
Под ссылочную переменную транслятор отводит место в памяти машины; эта переменная имеет имя и явно упоминается в программе. Ссылочные переменные образуют новый тип данных — «ссылки» (указатели).

Динамические переменные, как правило, имеют тип «запись» (RECORD), так как должны содержать, помимо значения (целого, вещественного и т. п.), ссылку на другую динамическую переменную связанной структуры.

Пример. Пусть в памяти машины имеется динамическая переменная, содержащая поле целого значения 2 и поле ссылки (указатель) на другую компоненту связанной структуры (цепочки):



Адрес данной переменной (ссылка) содержится в ссылочной переменной R.



Обозначим тип ссылочной переменной через POINT, а тип динамической переменной через CT. Тогда этот факт описывается следующим образом:

```
TYPE POINT= ↑CT;
```

Говорят, что «тип POINT указывает (ссылается) на компоненты типа CT» или «тип POINT связан с типом CT».

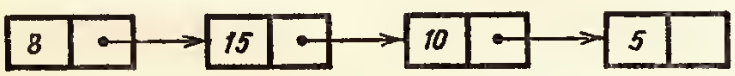
Ссылочную переменную R можно описать двумя способами:

```
а) TYPE POINT= ↑CT;   либо б) VAR R: ↑CT;
   VAR R:POINT
```

Переменная R указывает на компоненты типа CT.

Чтобы связать динамические переменные в цепочку, надо в каждой компоненте иметь ссылку на предыдущую компоненту.

Например, компоненты, содержащие числа 5,10,15,8, должны иметь еще информацию о том, где находится предыдущий элемент, так как это не массив и компоненты размещаются необязательно подряд.



Опишем тип таких данных, обозначив его CT. Очевидно, этот тип есть «запись» с двумя полями: полем целого значения (I) и полем

ссылки (P)

```
TYPE CT=RECORD
  I:INTEGER;
  P:POINT
  END;
```

Очевидно, ссылочная переменная, указывающая на такого типа данные, должна иметь тоже тип POINT. Опишем этот тип:

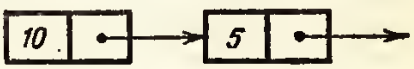
```
TYPE POINT= ↑CT;
```

Как мы видим, возник порочный круг: для описания типа POINT привлекается понятие CT, а при описании типа CT необходимо использовать POINT.

Условились в этом случае *сначала* описывать тип ссылочной переменной, а *затем* уже тип компоненты:

```
TYPE POINT= ↑CT;
  CT=RECORD
     I:INTEGER;
    P:POINT
  END;
```

Правила языка паскаль *только* при *описании ссылок* допускают использование идентификатора (CT) *до* его описания; во всех остальных случаях, прежде чем упомянуть идентификатор, необходимо описать его тип. Рассмотрим схему образования цепочки динамических данных, содержащих числа 5,10.

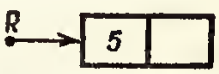


Машине необходимо произвести следующие действия:

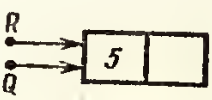
1. Найти и зарезервировать место в памяти для компоненты:

2. Заслать ссылку на эту компоненту (адрес) в ссылочную переменную R:

3. Присвоить полю I значение 5:



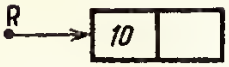
4. Присвоить некоторой ссылочной переменной Q значение R (скопировать):



5. Найти и зарезервировать место в памяти для новой компоненты:

6. Заслать в переменную R адрес этой компоненты:

7. Заслать в поле I значение 10:



8. Заслать в поле P значение Q:

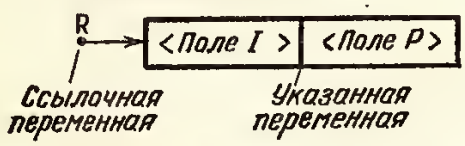
Последовательность подобных действий создает цепочку динамических переменных.

22.1. Процедура NEW

Резервирование места в памяти под динамическую переменную и засылка этого адреса в ссылочную переменную R выполняется при обращении NEW(R). При этом выделяется столько ячеек памяти, сколько требует динамическая переменная, с которой связана R. Эти все данные система получает из раздела описания типов в программе.

Динамические переменные, созданные посредством процедуры NEW(R), называют также *указанными переменными* (указатель R).

Пример. Пусть переменная R имеет тип POINT, описанный выше. Тогда после обращения к процедуре NEW(R) будет создана *указанная* переменная, в которой предусмотрено поле под значение типа INTEGER и поле ссылки. При этом ссылочная переменная R содержит адрес указанной переменной. Через R↑ обозначается сама указанная переменная; R↑. I — поле целого значения I, R↑.P — поле ссылки P.



22.2. Операции над ссылочными переменными

Значение ссылочной переменной R можно присваивать другой ссылочной переменной того же типа.

Пример 1. Пусть Q,R:↑POINT; тогда оператор Q:=R; заслет в Q тот же адрес, что хранится в R.

Рассмотрим действия со ссылочными переменными на следующей схеме. Пусть Q и R указывают на различные компоненты динамических переменных типа C:

```
C=RECORD
  I:INTEGER;
  P:POINT
  END;
```

Пусть в памяти машины размещены две цепочки динамических переменных (рис. 7):

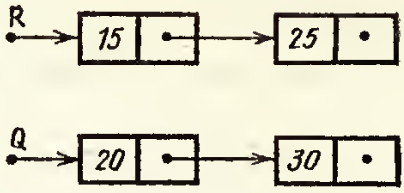
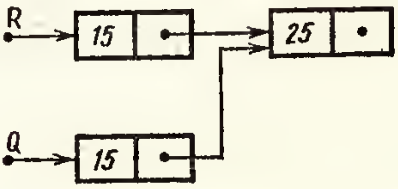


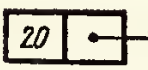
Рис. 7

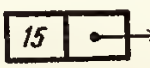
Выполним один из четырех операторов: Q:= R; Q↑:=R↑; Q↑.I:=R↑.I; либо Q↑.P:=R↑.P;

а) После выполнения оператора Q:=R; переменная Q указывает на ту же динамическую переменную, что и R:

б) После выполнения оператора Q↑:=R↑; (из исходного состояния (рис. 7)) получим

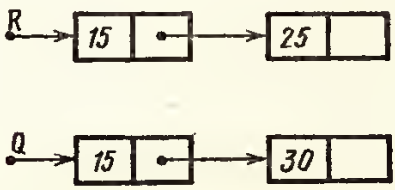


На место указанной переменной 

, указывавшей на 30, заслана переменная 

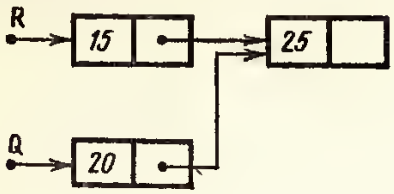
, указывающая на 25.

в) После выполнения оператора Q↑.I:=R↑.I из исходного состояния (рис. 7) получим следующее:

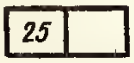


На место целого значения 20 заслано значение 15; поле указателя не изменилось.

т) После выполнения оператора Q↑.P:=P↑.P из исходного состояния (рис. 7) получим:



На место ссылки на компоненту  заслана ссылка на

компоненту  , поле целого значения не изменилось.

Ссылочные переменные могут указывать на одну и ту же переменную, т. е. быть равными, как R и Q в случае а).

Ссылочные переменные можно сравнивать посредством операций = и < >. Логическое выражение Q = R имеет значение TRUE для случая а) и значение FALSE для случаев б) и в), так как для б) ссылочные переменные Q и R указывают на *разные* динамические переменные, имеющие, правда, равные значения.

В качестве аналога нуля для ссылочных переменных принято специальное значение NIL: если переменная имеет значение NIL, то это означает, что она не указывает ни на какую переменную. Значение NIL в поле указателя имеет всегда *первая* компонента цепочки динамических переменных.

Пример. 

Значение NIL можно заслать оператором присваивания: L:=NIL; если L=NIL, то цепочка пуста.

Чтобы определить, что данный элемент является первым в цепочке переменных, достаточно проверить на NIL значение поля указателя этой переменной.

Пример. IF L↑.P=NIL THEN...

Замечание. Попытка обратиться к указанной переменной с указателем, имеющим значение NIL, приводит к ошибке. Диагностика в этом случае не всегда выдается.

22.3. Процедура DISPOSE

Динамическая переменная, созданная процедурой NEW, может быть «стерта» только процедурой DISPOSE.

Общий вид:

DISPOSE(R);

Здесь R — ссылочная переменная, указывающая на ту динамическую переменную, которую следует стереть. После стирания динамической переменной R↑ нельзя использовать значение R, такая ошибка может привести к порче памяти и другим серьезным последствиям.

Динамические переменные, не стертые посредством DISPOSE, продолжают занимать место в памяти после окончания работы соответствующего блока программы (становятся «мусором»). Поэтому необходимо все лишние динамические переменные стереть перед окончанием работы блока.

22.4. Стек («магазин»)

Начнем с рассмотрения примера. Пусть в трубку с запаянным концом закатывают шарики (рис. 8). Извлекать их можно только в обратном порядке: тот шарик, что закатился последним, будет извлечен первым.

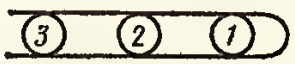


Рис. 8

Подобным образом можно организовать и данные. *Стек* — такая структура динамических данных, которая состоит из переменного числа компонент одинакового типа. Компоненты извлекаются из стека таким образом, что *первой* выбирается та компонента, которая была помещена *последней*. Извлеченная компонента в стеке не сохраняется.

Пример. Рассмотрим последовательные этапы засылки в стек чисел 1,2,3 (рис. 9).

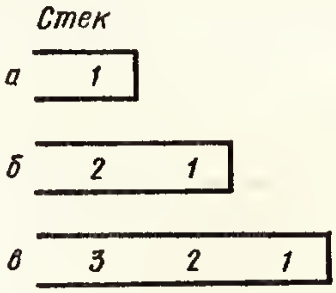


Рис. 9

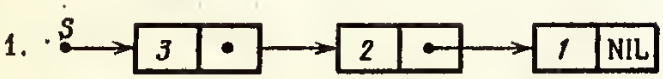
На этапе б) обращение к процедуре извлечения из стека дает число 2, на этапе в) — число 3.

Опишем стек, в который можно помещать цепочку динамических переменных:

```
TYPE STACKP=↑STACKCOMP;
STACKCOMP=RECORD
```

```
    I:INTEGER;
    P:STACKP
    END;
VAR S:STACKP;
```

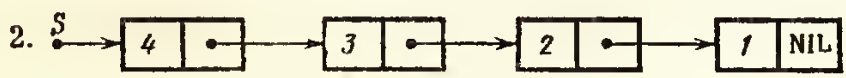
Если поместить в этот стек последовательно числа 1,2,3, то получится следующий вид:



Поместить в такой стек компоненту можно, например, процедурой SN:

```
S:=NIL;
. . . .
PROCEDURE SN(K:INTEGER);
VAR INEW:STACKP;
(* ТИП STACKP ДОЛЖЕН БЫТЬ ОПИСАН ВЫШЕ,
НАПРИМЕР, В PROGRAM *)
BEGIN
(* РЕЗЕРВИРУЕТСЯ ПАМЯТЬ ПОД НОВУЮ КОМПОНЕНТУ, А В INEW ЗАСЫЛАЕТСЯ АДРЕС ЭТОЙ КОМПОНЕНТЫ *)
NEW (INEW);
  WITH INEW DO
    BEGIN I:=K; P:=S END;
  S:=INEW;
```

Если со стеком вида 1) обратиться к процедуре SN для засылки числа 4, то получим стек вида 2):



Процедура извлечения компоненты из такого стека может иметь следующий вид:

```
PROCEDURE OUT(VAR K:INTEGER);
VAR IOLD:STACKP;
BEGIN
  IOLD:=S;
(* АДРЕС ПОСЛЕДНЕЙ КОМПОНЕНТЫ *)
  K:=IOLD↑.I;
```

```
(* ИЗВЛЕКАЕТСЯ И ЗАСЫЛАЕТСЯ В S ЗНАЧЕНИЕ
СООТВЕТСТВУЮЩЕГО УКАЗАТЕЛЯ НА 3 *)
  S:=IOLD ↑ .P;
  DISPOSE (IOLD)
END;
```

После обращения к процедуре OUT стек вернется к виду 1).

Пустым стеком называется стек, не содержащий компонент. Такой стек можно получить, присвоив значение NIL соответствующей ссылочной перемепной (в нашем случае S:=NIL;).

Если к пустому стеку применить несколько раз процедуру SN, а затем *столько же* раз процедуру OUT, то получим снова пустой стек.

З а м е ч а н и е. Нельзя применять процедуру OUT к пустому стеку.

Стеки позволяют гибко и экономно использовать память, так как в каждый момент в стеке могут находиться только те переменные, которые нужны для дальнейшей работы программы. (В то время как под массивы, например, мы часто вынуждены резервировать и держать избыточную память).

22.5. Очередь

Очередь — такая структура данных, при которой *изъятие* компонент происходит *из начала* цепочки, а *запись* — *в конец* цепочки.

В этом случае вводят два указателя: один на начало очереди, другой — на ее конец.

22.6. Дозапись новых компонент

П р и м е р. Пусть имеется цепочка дипамических переменных (рис. 10).

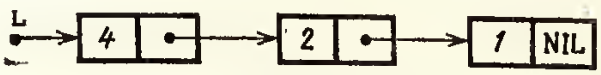
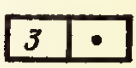
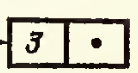


Рис. 10

Переменные имеют описанный выше тип STACKCOMP

Требуется вставить в цепочку новую компоненту 

после компоненты 

, если известен указатель NEWP 

.

Для записи этой новой компоненты достаточно выполнить операторы:

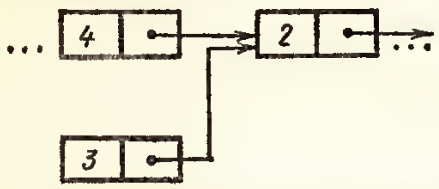
```
NEWP↑.P:=L↑.P;
L↑.P:=NEWP;
```

Первый оператор засылает в поле указателя повой компоненты 

ссылку на компоненту 

. Эта ссылка находится в поле указателя последней компоненты: 

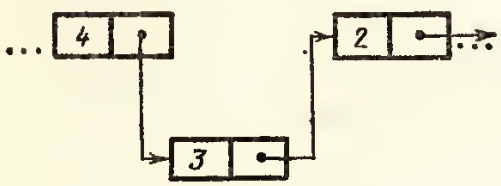
, т. е. получается следующий вид:



Второй оператор помещает в поле указателя компоненты 

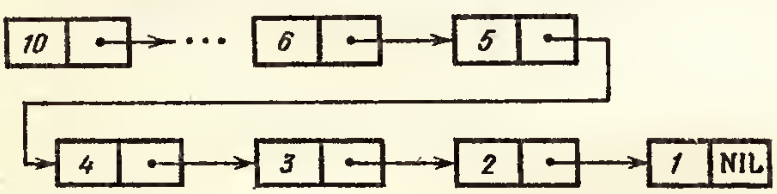
ссылку на компоненту 

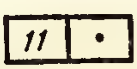
. Получается следующая картинка:

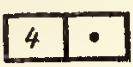


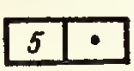
З а д а ч а. Построить цепочку динамических переменных, содержащих целые числа, а затем между 4-й и 5-й переменной вставить новую динамическую переменную.

Пусть требуется построить такую цепочку:



и вставить элемент 

между 

и 

Р е ш е н и е:

```
PROGRAM POINT (INPUT,OUTPUT);
TYPE INTP=↑INTREC;
  INTREC=RECORD
```

```
    I:INTEGER;
    P:INTP
  END;
VAR IP,IR,INSERTI, INSERT4,INSERT5,WRTP:INTP;
  N,K,L,M:INTEGER;
BEGIN
  IR:=NIL;N:=10;
  FOR K:=1 TO N DO
  BEGIN READ(L); WRITE (' ',L);
    NEW (IP); IP↑.I:=L; IP↑.P:=IR; IR:=IP;
    IF K=5 THEN
    BEGIN INSERT5:=IP; INSERT4:=IP↑.P END
  END;
WRITELN;
READ(L); WRITELN(' L=',L);
NEW(INSERTI); INSERTI↑.I:=L;
INSERTI↑.P:=INSERT4;
INSERT5↑.P:=INSERTI;
FOR K:=1 TO N + 1 DO
  BEGIN WRTP:=IR; M:=WRTP↑.I; WRITE ('_', M);
    IR:=WRTP↑.P;
    DISPOSE(WRTP)
  END;
WRITELN
END.
```

Результат:

```
 1 2 3 4 5 6 7 8 . . . 10
L = 11
 10 9 8 7 6 5 11 4 . . . 1
  9
  8
  7
  6
  5
 11
  4
  3
  2
  1
```

Вводятся числа: 1; 2; 3; 4; 5; 6; 7; 8; 9; 10 и 11. Между 4 и 5 вставляется число 11.

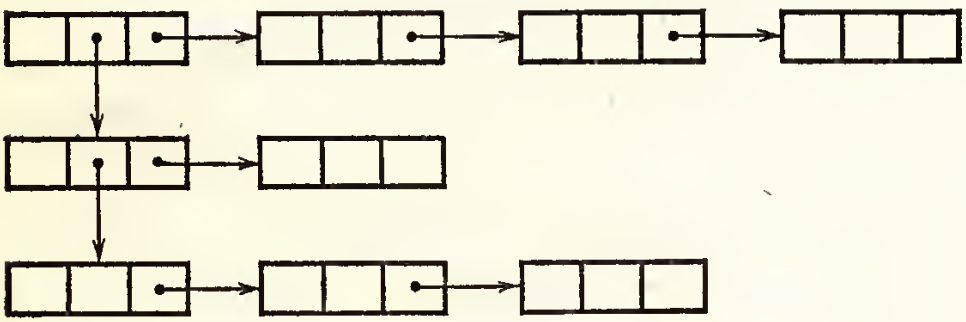
22.7. Нелинейные структуры

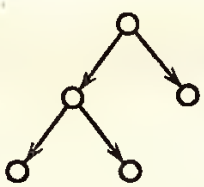
До сих пор мы рассматривали *линейные* структуры динамических переменных.

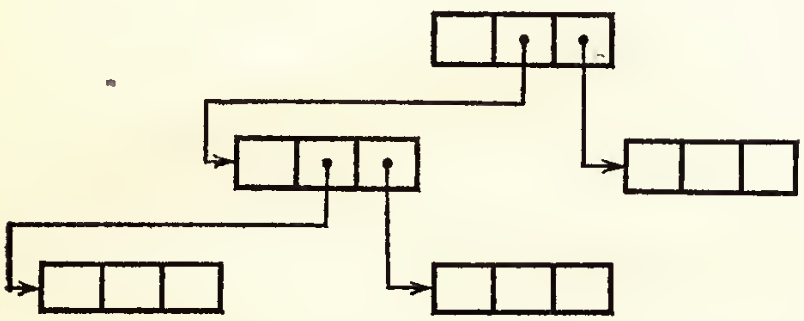
Введение в динамическую переменную двух и более полей указателей создает возможность получать *нелинейные структуры*.

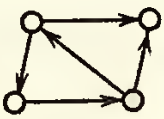
Примеры нелинейных структур показаны ниже.

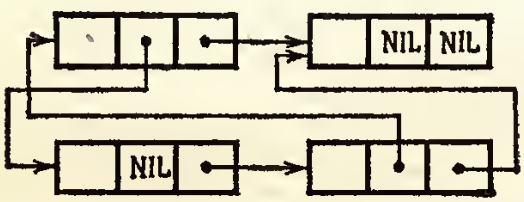
а) Текст



б) Двоичное дерево  можно представить так:



в) Направленный граф  можно представить так:



а) Текст — это связанная структура. Ее элементы представляют собой записи с тремя полями: первое поле содержит текстовую информацию (например, слово), второе поле либо указывает на пер-

вый элсмент следующей строкн, либо имеет значение NIL, третье — на слсдующий элемент данной строки.

б) Компононта двоичного дерева также имеет три поля: первое поле содержит основную информацию (число, символ, слово и т. д.), а второе и третье поля содержат ссылки на предыдущую и последующую компоненты дерева. Для некоторых элементов дерева одно из этих полей либо оба имеют значение NIL.

в) Посредством ссылок можно выразить все связи в структуре, моделирующей направленный граф: вершинам графа соответствуют сами динамические переменные, а ребрам — ссылкн.

Узлом называют переменную, содержащую два различных, отличных от NIL, значения указателя. Так, узловые элементы текста содержат ссылкн на очередной элемент данной строки и на первый элемент следующей строки.

Нелинейные структуры удобны для задач поиска. Список упорядоченных элементов в виде двоичного дерева представлен на рис. 11. Узсл 4 называется *корнем* дерева. Вход в дерево происходит только через корень. Для каждого узла различаются левое и правое *поддеревья*. Упорядоченность элементов следующая: элементы левого поддерева меньше узла и меньше элементов правого поддерева.

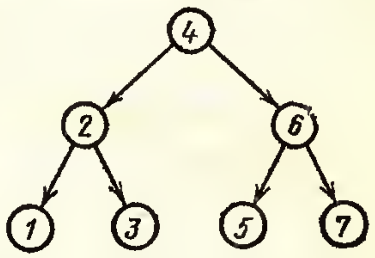


Рис. 11

Время поиска в двоичном дереве сокращается по сравнению с линейной структурой с $\sim N$ до $\sim \log_2 N$, где N — число узлов в дереве. Компоненту двоичного дерева можно представить переменной типа

```
BIREC=RECORD
  I:INTEGER;
  PRED,SUCC:INTP
  END;
```

Каждая такая переменная содержит три поля: поле целого значения — поле I, поле указателя на предыдущий (в смысле упорядоченностн) элемент — поле PRED и поле указателя на последующнй элемент — поле SUCC.

23. Работа с внешними модулями *)

Паскаль позволяет работать с внешнимн процедурами (функциямн), которые существуют вне главной программы (PROGRAM).

*) Работа с внешнимн модулями — расшнрение стандарта языка паскаль.

Если модуль является стандартным (библиотечным), то никаких описаний его в программе не требуется.

В остальных случаях внешний модуль должен быть описан в PROGRAM следующим образом:

1. Процедура паскаля:

PROCEDURE N(P1:T1;...);EXTERNAL;

Здесь N — имя процедуры, P1, . . . — формальные параметры, T1, . . .— типы формальных параметров (напоминаем, что параметру, предназначенному для результата, должно предшествовать ключевое слово VAR).

2. Функция паскаля:

FUNCTION F(X:TYPEX; . . .):TYPEF; EXTERNAL;

Здесь F — имя функции, X — формальный параметр, TYPEX — тип этого параметра, TYPEF — тип результата, EXTERNAL (EXTERN) — указание на то, что эта функция существует вне данной программы.

Замечание. На БЭСМ-6 следует писать EXTERNAL, на ЕС ЭВМ — EXTERN;

3. Подпрограмма фортрана:

PROCEDURE SUB (X:TYPEX; ...; FORTRAN;

Здесь X — параметр, TYPEX — его тип, FORTRAN — указание на то, что использован фортранный модуль.

4. Подпрограмма-функция фортрана:

FUNCTION F(X:TYPEX;...):TYPEF; FORTRAN;

Ниже приводится пример, который поможет читателю использовать внешние модули в своих программах.

Пример. (ЕС ЭВМ).

```
//TEST5 JOB C3746, SEMASHKO,MSGLEVEL=(2,0)
// _EXEC FORTGC
//FORT. SYSIN DD *
      SUBROUTINE SUB(X)
C ТРАНСЛЯЦИЯ SUB
      X=2.
      RETURN
      END
      FUNCTION F(X)
C ТРАНСЛЯЦИЯ F
      F=4*X
      RETURN
      END
//_EXEC PASC
```

```
//PASC.SYSIN DD *
PROGRAM FILLER (OUTPUT);
(* ТРАНСЛЯЦИЯ PROC *)
(* ¤E+ *)
PROCEDURE PROC (VAR X:REAL);
  BEGIN X:=1.0 END;
BEGIN END.
// _EXEC PASC
//PASC.SYSIN DD *
PROGRAM FILLE(OUTPUT);
(* ТРАНСЛЯЦИЯ FUNPSCL *)
(* ¤E+ *)
FUNCTION FUNPSCL(X:REAL):REAL;
  BEGIN FUNPSCL:=3.0_END;
BEGIN END.
//_EXEC PASCLG,
//PASC.SYSIN DD *
PROGRAM GETHER (OUTPUT);
(* ТЕСТ НА ВНЕШНИЕ МОДУЛИ *)
VAR A,B:REAL;
  PROCEDURE SUB(VAR X:REAL); FORTRAN;
  FUNCTION F(X:REAL):REAL; FORTRAN;
  FUNCTION FUNPSCL(X:REAL):REAL; EXTERN;
  PROCEDURE PROC (VAR X:REAL); EXTERN;
BEGIN A:=1.0;
  SUB(B); WRITE(' SUB(B):' ,B, 'F(A)= ',F(A));
  A:=FUNPSCL(A); WRITELN(' A=',A);
  PROC(B);WRITELN(' PROC(B):',
  B,'FUNPSCL(A)=',A)
END.
//LKED.SYSLIB DD
// DD DSN=SYS1.FORTLIB,DISP=SHR
//GO.SYSIN DD *
//
```

Результат:

```
SUB(B):2.000...F(A)=4.000...A=3.000...
PROC(B):1.000...FUNPSCL(A)=3.000...
```

В этом примере выполняется программа GETHER, использующая внешние фортранные модули SUB и F, а также модули паскаля PROC и FUNPSCL, описанные вне программы GETHER.

Внешние процедуры и функции не должны иметь в качестве параметров параметры-функции и сами не могут быть переданы в качестве параметров.

Внешняя процедура может использоваться рекурсивно и передана как параметр только процедуре, декларированной *внутри* нее самой.

23.1. Трансляция внешних модулей

Необходимо соблюдать следующие правила.

1. Внешние паскаль-процедуры и функции транслируются только в режиме E+ (см. п. 24).

2. Режим E+ должен быть установлен до декларации внешней процедуры.

В этом режиме не должны транслироваться внутренние процедуры и главная программа.

3. Допускается только два уровня вложения внешних процедур. Однако сама внешняя процедура может иметь несколько уровней вложения внутренних (для нее) процедур.

4. Если во внешней процедуре описываются глобальные переменные, константы или типы, то они должны в точности совпадать по описанию с соответствующими переменными, константами, типами в PROGRAM.

5. Если однажды установлен режим E+, то нельзя его устанавливать повторно.

Когда паскаль-программа вызывает фортранную подпрограмму или функцию, надо учитывать следующее.

1. Тип REAL на ЕС ЭВМ соответствует фортраннопу DOUBLE PRECISION (или REAL*8).

2. Тип BOOLEAN соответствует фортранному LOGICAL.

3. Массивы паскаля располагаются по строкам, а массивы фортрана — по столбцам.

З а м е ч а н и е. При передаче двумерного массива в качестве параметра из паскаль-процедуры в фортранную (и обратно) этот массив необходимо транспонировать.

24. Режимы трансляции

Режимы трансляции задаются комментариями особого вида (псевдокомментариями):

(* CH R1,R2,... *)

Здесь (* и *) — ограничители, CH — специальный символ, превращающий комментарий в управляющую карту, R1,R2 — задаваемые коды режимов трансляции. На БЭСМ-6 CH — это символ «=», на ЕС ЭВМ CH — это символ «$» (либо совпадающий с ним по кодировке знак ¤).

Каждый код режима состоит из двух символов: буквы, за которой следует либо знак («+» или «—»), либо цифра.

Знак «+» означает включение данного режима, знак «—» отказ от него.

Пример.

(* =T+,E+,P— *) либо (* ¤T+, E+, P— *)

Чаще всего используются следующие режимы:

1. T — этот режим обеспечивает динамические проверки во время счета:

а) всех операций с индексными переменными на принадлежность каждого индекса допустимому диапазону индексов;

б) всех операторов присваивания на принадлежность значений переменных ограниченного типа соответствующему подмножеству;

в) всех делителей в операциях деления (на нуль);

г) всех автоматических преобразований

INTEGER → REAL на ABS(I) < =MAXINT;

д) всех операторов CASE на соответствие переключателя одной из меток CASE.

По умолчанию установлен T+.

Для отлаженных программ рекомендуется использовать T—. Это ускоряет выполнение программы.

2. P позволяет выдавать подробную информацию при «авостах» (аварийных остановах — прекращении счета при ошибке). В режиме P+ выдаются значения локальных переменных, идентификаторы вызванных процедур (функций) и номера строк программы, в которых начинаются соответствующие составные операторы.

По умолчанию установлен P+.

Для отлаженных программ следует указывать режим P—, что экономит память и время выполнения программы.

3. E позволяет так транслировать процедуры и функции, что к ним можно обращаться из других программ как к внешним модулям.

Если данная процедура используется как EXTERNAL, то ее необходимо транслировать только в режиме E+.

По умолчанию установлен E—.

4. U+ все символы входной строки, начиная с 73-го, считаются комментариями. На БЭСМ-6 режима U нет. Если используется U—, то все символы, начиная со 121-го, считаются комментариями.

5. BN — для БЭСМ-6. Пусть S — нижняя граница размера памяти, выделенной под буфер файлов, S > 256*N.

По умолчанию установлено N=1.

B+ для ЕС ЭВМ — зарезервированные паскалем ключевые слова (AND,ARRAY,...,WITH) на листинге АЦПУ печатаются жирно.

По умолчанию установлен B—.

6. L управляет информацией об исходной программе, выдаваемой на АЦПУ.

На ЕС ЭВМ:

L+ — подробная выдача,

L— — режим счета, подавление листинга программы.

По умолчанию установлен L+.

На БЭСМ-6:

L0 — выдаются только сообщения об ошибках (*NO LIST);

L1 — выдается таблица загрузки и текст программы;

L2 — дополнительно к информации, выдаваемой по L1, выдаются коды стандартного массива (см. [7]).

По умолчанию установлен L1.

ПРИЛОЖЕНИЕ 1

ПРОГРАММА ИЗМЕНЕНИЯ ДЛИНЫ СТРОК ТЕКСТА

(пример использования записей с вариантами)

З а д а ч а. Составить программу, вводящую текстовую информацию со строками длиной до 80 символов и выдающую этот же текст со строками длиной до 60 символов (без разбиения слов при переносе).

В тексте могут присутствовать пустые строки; абзац начинается с нескольких пробелов. Абзацем будем считать любую непустую строку, имеющую три и более пробелов в начале. Конец всего текста пусть будет отмечен специальным символом EOT, конец строки — символом EOL.

Тогда любой текст можно представить в виде последовательности элементов: слов, знаков препинания и символов EOL и EOT. Все элементы разделяются пробелами. Простейшая схема программы приводится на рис. 12.

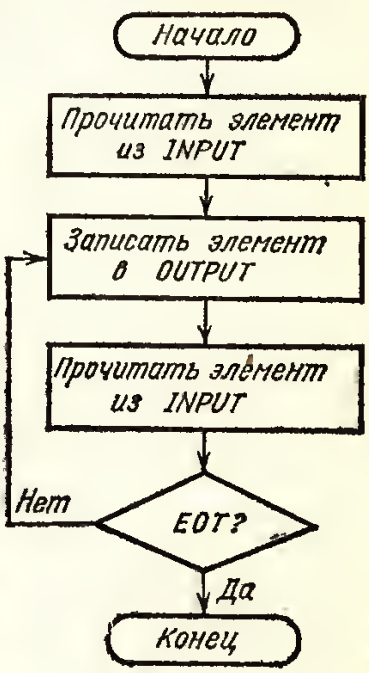


Рис. 12

Напомним, что слова не разбиваются для переноса: если слово умещается, оно записывается в данную выходную строку; если не умещается, то слово целиком записывается в следующую строку OUTPUT. Элемент «слово» и элемент «знак» неравнозначны: знак обязательно должен стоять в той же строке, что и слово, за которым он поставлен, а последовательные слова можно писать в разных строках.

Поэтому программа после ввода каждого элемента должна ввести еще один элемент и только после его анализа обрабатывать предыдущий (оставлять на текущей строке либо переносить на следующую).

Усложненная схема программы, имя которой INOUT, приводится на рис. 13.

Пусть чтение элемента выполняется процедурой RNEXT и запись — процедурой WTHIS.

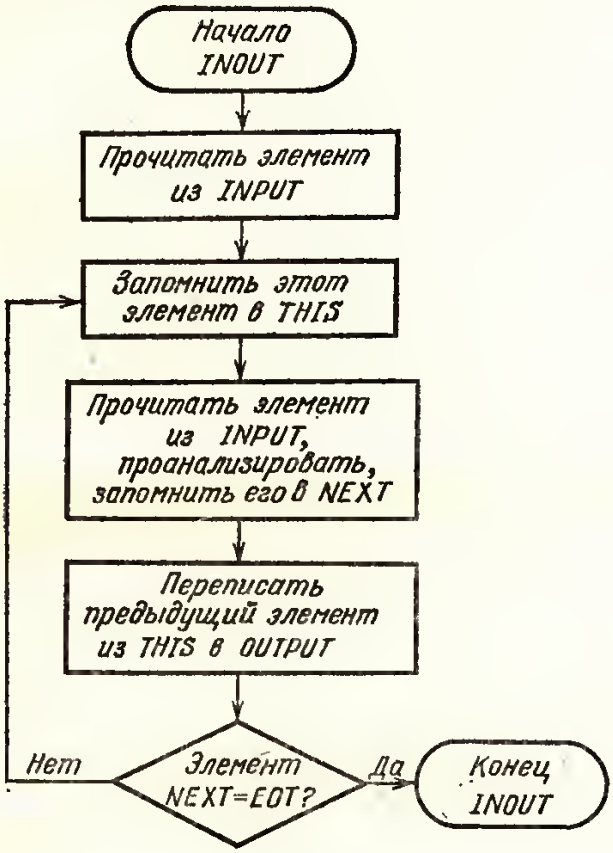


Рис. 13

Если в исходном тексте есть абзац или пустая строка, то будем считать, что в соответствующем месте текста есть некоторый элемент, играющий роль управляющего символа (CONTR).

Тогда весь текст можно себе представить состоящим из следующих элементов: слов (WORD), знаков препинания (PUNCT), управляющих символов (CONTR) и признака «конец текста» (EOT).

Будем считать, что слово состоит не более чем из 16 символов.

В этих предположениях произвольный элемент (ITEM) текста описывается как запись с вариантами:

```
TYPE ITEM=(WORD,PUNCT,CONTR,EOT);
  TEXTITEM=RECORD
    CASE KIND:ITEM OF
    WORD:(L:1..16;
      SP:ARRAY[1..16] OF CHAR);
    PUNCT:(P: CHAR);
```

```
CONTR:(C:(JUMP,ABZ));
EOT:( )
END;
```

Если текущий элемент есть «слово» (WORD), то задается его длина L и массив SP на 16 символов, где размещаются буквы этого слова; если элемент — «знак» (PUNCT), то в P находится соответствующий символ; если элемент — «управляющий символ» (CONTR), то в C находится признак «пустая строка» (JUMP) либо «абзац» (ABZ). Символ «конец текста» (EOT) служит только признаком конца и в выходной текст не попадает.

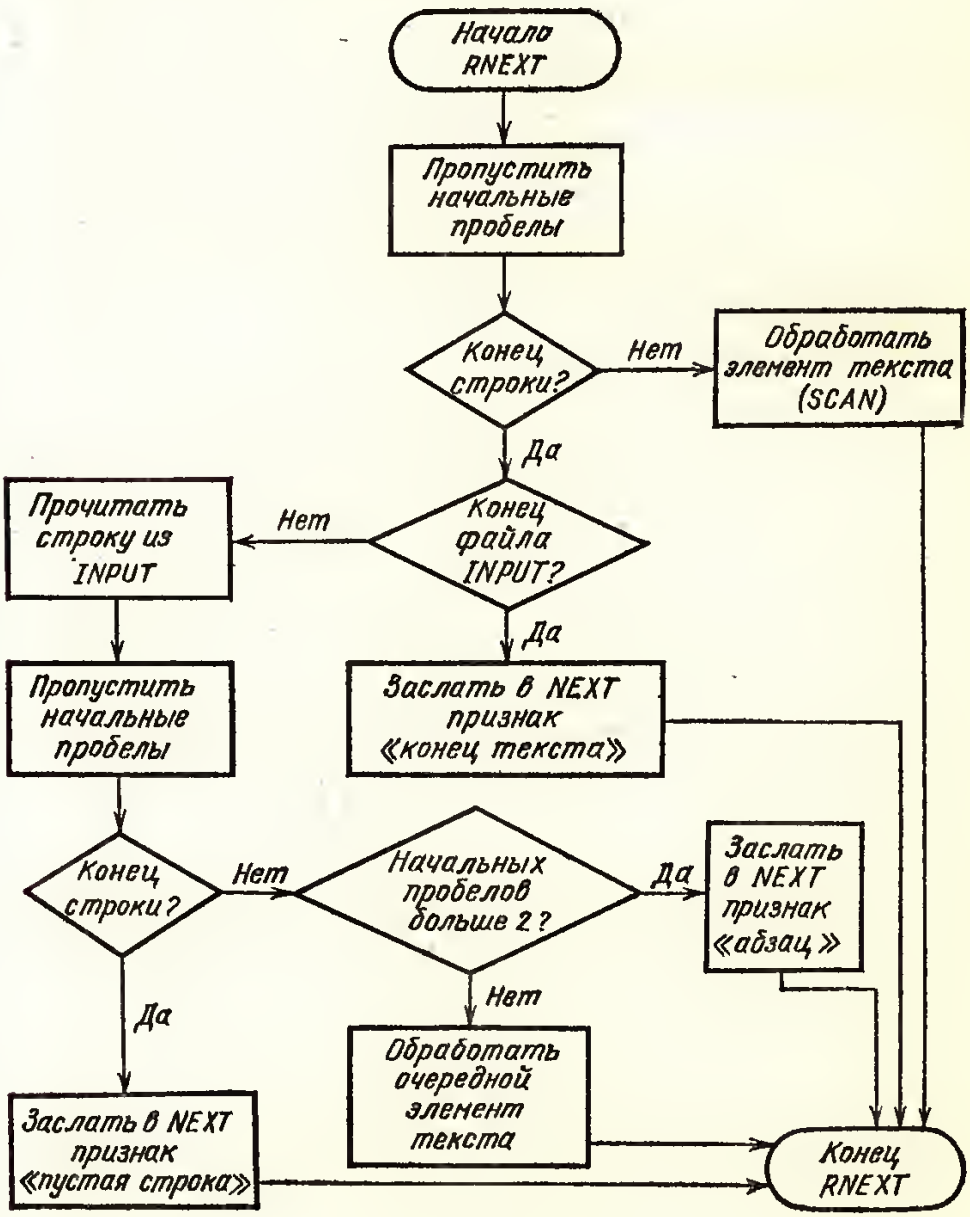


Рис. 14

Введем две переменные THIS и NEXT для хранения элемента текста

```
VAR THIS,NEXT:TEXTITEM
```

Процедура RNEXT должна прочитать символы, составляющие следующий элемент входного текста, проанализировать и поместить информацию в переменную NEXT, т. е. заполнить соответствующее поле записи типа TEXTITEM для переменной NEXT.

Удобнее вводить из INPUT не по одному элементу текста, а по целой строке. Для этого заведем массив LINE длиной от 1 до INMAX символов по числу символов во входной строке:

```
LINE:ARRAY [1..INMAX] OF CHAR
```

В нашем конкретном случае входная строка содержит до 80 символов. Для упрощения алгоритма программы будем отмечать конец входной строки каким-либо специальным символом, например «!». Этот символ поместим в LINE после последнего отличного от пробела символа введенной строки. Таким образом, длина массива LINE должна быть на 1 больше длины входной строки, т. е. для входной строки в 80 символов INMAX=81.

Элементы строки процедура выбирает из массива LINE. Схема RNEXT приводится на рис. 14.

Задачу «обработать элемент текста» можно оформить в виде процедуры SCAN (рис. 15).

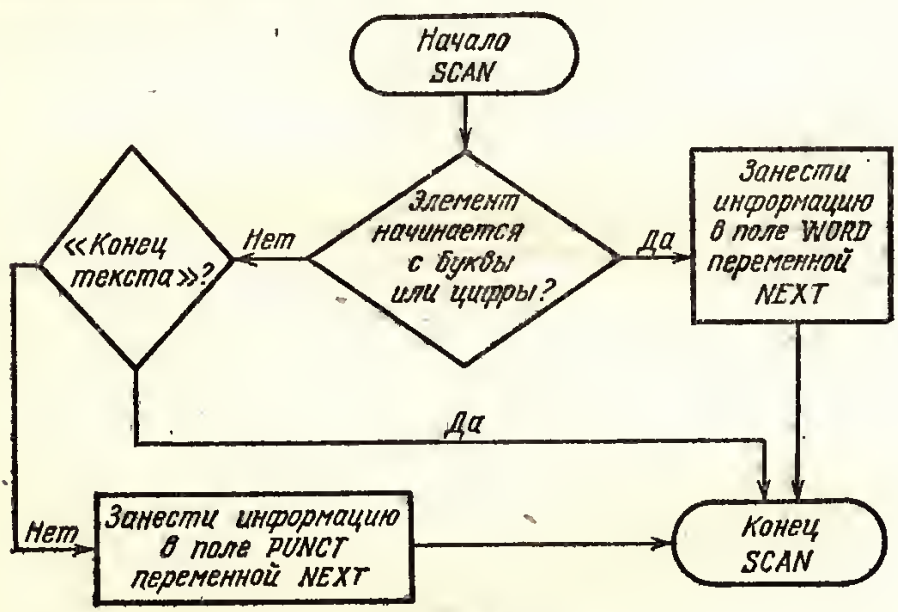


Рис. 15

На последнем этапе работы программы потребуется формировать выходные строки текста в файле OUTPUT. Запись элемента текста в выходную строку можно осуществить следующей процедурой WTHIS (рис. 16).

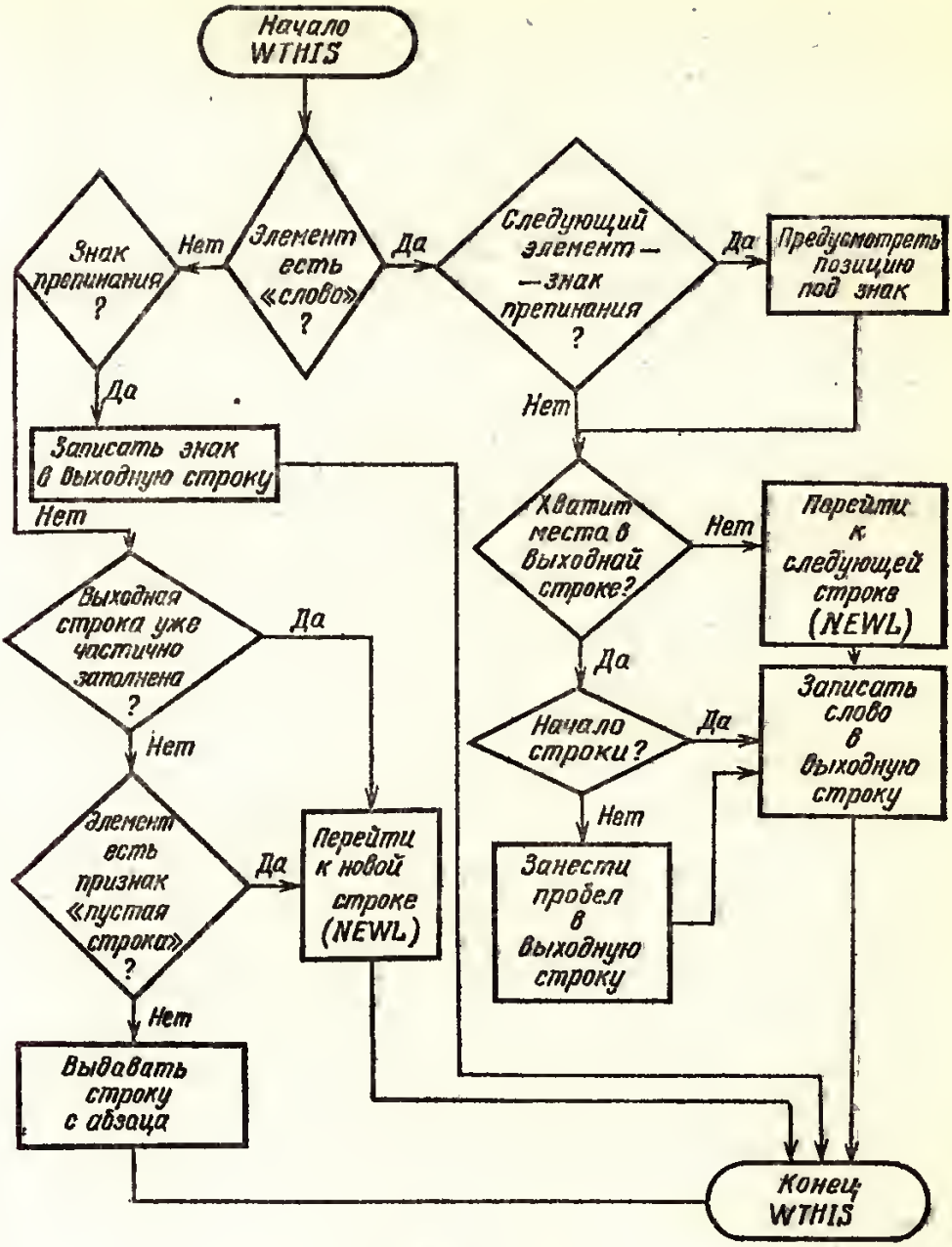


Рис. 16

Приведем полный текст программы. Здесь

INMAX — максимальная длина входной строки плюс единица;
OUTMAX — максимальная длина выходной строки;
EOT — признак конца текста;
EOL — признак конца строки;
LLINE — длина текущей входной строки;
THIS — предыдущий элемент текста;
NEXT — последующий элемент текста;
LINE — массив для хранения входной строки;

POS — номер текущей позиции в строке;
FREE — количество оставшихся свободных позиций в строке;
K — номер символа в слове;
SP — массив для хранения слова;
SPACE — число позиций, требуемое под очередное слово в выходной строке.

```
    PROGRAM INOUT (INPUT,OUTPUT);
    (* ПРЕОБРАЗОВАНИЕ 80-СИМВОЛЬНОЙ СТРОКИ В 60-
    СИМВОЛЬНУЮ *)
CONST INMAX=81;
    EOL='!';
    EOT='*';
    OUTMAX=60;
TYPE ITEM=(WORD,PUNCT,CONTR,EOT);
    TEXTITEM=RECORD
      CASE KIND:ITEM OF
        WORD:(L:1..16;
          SP:ARRAY (.1..16.)
            OF CHAR);
        PUNCT:(P:CHAR);
        CONTR:(C:(JUMP,ABZ));
        EOT:( )
      END;
    LLINE=1..INMAX; (* ДЛИНА ВХ. СТРОКИ *)
VAR THIS, NEXT:TEXTITEM;
    LINE:ARRAY(.LLINE.) OF CHAR;
    POS:LLINE; (* НОМЕР ПОЗИЦИИ В СТРОКЕ *)
    FREE:0..OUTMAX;
PROCEDURE RNEXT;
    (* ЧТЕНИЕ ЭЛЕМЕНТА ИЗ LINE *)
PROCEDURE RLINE;
    (* ВВОД СТРОКИ ИЗ INPUT *)
VAR M:LLINE;
BEGIN (* RLINE *)
    M:=1;
    WHILE NOT EOLN(INPUT) DO
      BEGIN READ(LINE(.M.));M:=M+1 END;
      READLN;
      LINE(.M.):=EOL; POS:=1
END; (* RLINE *)
PROCEDURE SCAN;
    (* ОБРАБОТКА ЭЛЕМЕНТОВ — 'СЛОВО'
       И 'ЗНАК ПРЕПИНАНИЯ' *)
VAR K:0,.16;
    ST:SET OF CHAR;
```

```
BEGIN (* SCAN *)
    ST:=(.'.',',',';',':',' '.);
    WITH NEXT DO
    CASE LINE (.POS.) OF
      'А','Б','В','Г','Д','Е','Ж','З',
      'К','Л','М','Н','О','П','Р','С',
      'У','Ф','Х','Ц','Ч','Ш','Щ','''','Ы',
      'Э','Ю','Я','И','Т','Ь','R',
      '0','1','2','I','D','3','F','G','4',
      'J','5','L','6','N','7','8','Q',
      'S','9','U','V','W','—','Y','Z':
        BEGIN (* НАЧИНАЕТСЯ С БУКВЫ *)
          KIND:=WORD; K:=0;
          REPEAT K:=K+1;
            SP(.K.):=LINE(.POS.);
            POS:=POS+1
          UNTIL (LINE(.POS.) IN ST);
          L:=K
        END;
      '.',',',';',':',':':
        BEGIN (* ЗНАК ПРЕПИНАНИЯ *)
          KIND:=PUNCT;
          P:=LINE(.POS.);
          POS:=POS+1
        END;
      '*':KIND:=EOT (* КОНЕЦ ТЕКСТА *)
    END (* CASE *)
END; (* SCAN *)
BEGIN (* RNEXT *)
    WHILE LINE(.POS.)=' 'DO POS:=POS+1;
    IF LINE (.POS.)=EOL
    THEN IF EOF (INPUT)
         THEN NEXT.KIND:=EOT
         ELSE BEGIN RLINE;
             WHILE LINE (.POS.)='
             DO POS:=POS+1;
             WITH NEXT DO
               IF LINE(.POS.)=EOL
               THEN BEGIN
                 KIND:=CONTR;
                 C:=JUMP
                 END
               ELSE IF POS > 2
                 THEN BEGIN
                   KIND:=CONTR;
```

```
                    C:=ABZ
                    END
                  ELSE SCAN
              END
    ELSE SCAN
END; (* RNEXT *)
PROCEDURE WTHIS;
(* ЗАПИСЬ THIS В OUTPUT *)
VAR SPACE:1..18;
    M:1..16;
    PROCEDURE NEWL;
    (* ПЕРЕХОД К НОВОЙ ВЫХ. СТРОКЕ *)
    BEGIN
      WRITELN;
      WRITE(' ');
      FREE:=OUTMAX-1
    END; (* NEWL *)
BEGIN (* WTHIS *)
    WITH THIS DO
      CASE KIND OF
        WORD:BEGIN (* 'СЛОВО' *)
          IF NEXT.KIND=PUNCT
          THEN SPACE:=L+2
          ELSE SPACE:=L+1;
          IF SPACE > FREE
          THEN NEWL
          ELSE IF FREE < > OUTMAX
              THEN BEGIN
              WRITE (' ');
              FREE:=FREE-1
            END;
          FOR M:=1 TO L
            DO WRITE (SP(.M.));
          FREE:=FREE-L
          END;
        PUNCT:BEGIN (* ЗНАК ПРЕПИНАНИЯ *)
          WRITE(P); FREE:=FREE-1
          END;
        CONTR:BEGIN (*'ПРОПУСК СТРОКИ' ЛИБО 'АБЗАЦ' *)
          IF FREE<OUTMAX
          THEN NEWL;
          IF C=JUMP
              THEN NEWL
              ELSE BEGIN
               WRITE (' ');
```

```
            FREE:=FREE-2
            END
          END;
        EOT:BEGIN (* КОНЕЦ ТЕКСТА *)
          FOR M:=1 TO L DO
            WRITE (SP(.M.));
          FREE:=FREE-L
          END
      END (* CASE *)
END; (* WTHIS *)
BEGIN (* INOUT *)
    LINE(.1.):=EOL; POS:=1;
    FREE:=OUTMAX;
    RNEXT;
    WRITE ('_'); FREE:=FREE-1;
    REPEAT
      THIS:=NEXT; (* ПЕРЕПИСЬ ЭЛЕМЕНТА
                     ТЕКСТА В THIS *)
      RNEXT;
      WTHIS (* ВЫДАЧА THIS В ВЫХ. СТРОКУ *)
    UNTIL NEXT.KIND=EOT;
END. (* INOUT *)
```

ПРИЛОЖЕНИЕ 2

ПРОГРАММА РЕШЕНИЯ ЗАДАЧИ О ХАНОЙСКОЙ БАШНЕ

(пример использования рекурсивной процедуры)

В одной из древних легенд говорится следующее. «В храме Бенареса находится бронзовая плита с тремя алмазными стержнями. На один из стержней бог при сотворении мира нанизал 64 диска разного диаметра из чистого золота так, что наибольший диск лежит на бронзовой плите, а остальные образуют пирамиду, сужающуюся кверху. Это — башня Брамы. Работая день и ночь, жрецы переносят диски с одного стержня на другой, следуя законам Брамы:

1) диски можно перемещать с одного стержня на другой только по одному;

2) нельзя класть бóльший диск на меньший.

Когда все 64 диска будут перенесены с одного стержня на другой, и башня, и храмы, и жрецы-брамины превратятся в прах и наступит конец света».

Эта древняя легенда породила задачу о Ханойской башне: переместить m дисков с одного из трех стержней на другой, соблюдая «законы Брамы».

Назовем стержни левым (LEFT), средним (MIDDLE) и правым (RIGHT). Задача состоит в переносе m дисков с левого стержня на правый (рис. 17).

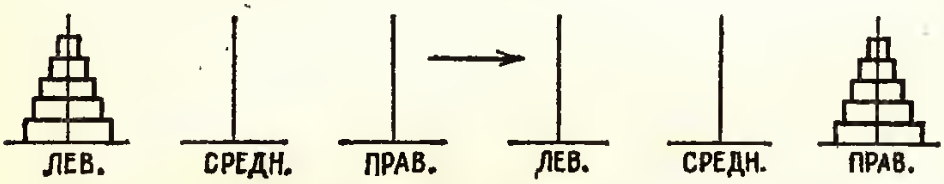


Рис. 17

Задача может быть решена одним перемещением только для одного (m = 1) диска. В общем случае потребуется 2^{m-1} перемещений.

Построим рекурсивное решение задачи, состоящее из трех этапов:

а) перенести башню, состоящую из (m — 1) диска, с левого стержня на средний (рис. 18):

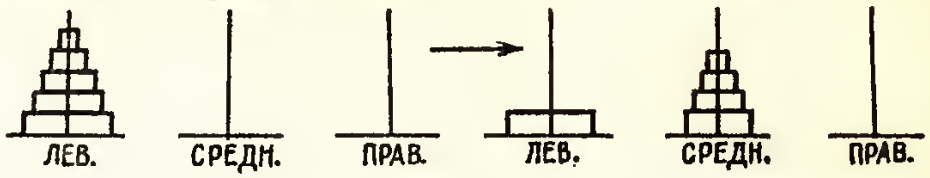


Рис. 18

б) перенести один оставшийся диск с левого стержня на правый (рис. 19):

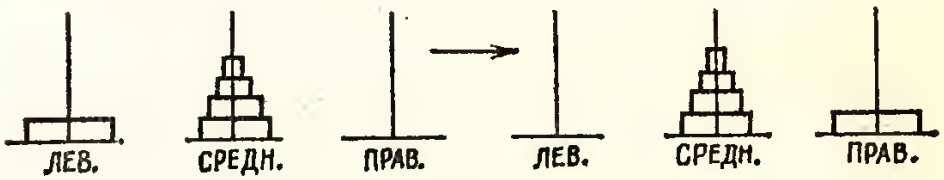


Рис. 19

в) перенести башню, состоящую из (m — 1) диска, со среднего стержня на правый (рис. 20):

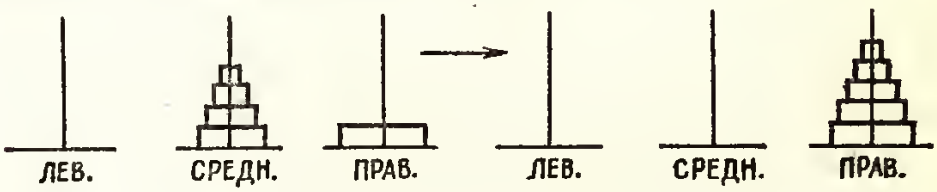


Рис. 20

Таким образом, задача о перемещении m дисков сводится к задаче о перемещении (m — 1) диска. Обращаясь опять к этому же алгоритму, сведем задачу к перемещению (m — 2) дисков. Продолжая этот процесс, получим в конце концов задачу о перемещении одного диска. Эта задача решается за один ход. Таким образом, в процессе решения возникают промежуточные задачи: переместить башню из нескольких (n) дисков с одного стержня на другой.

Обозначим тот стержень, с которого следует снять диски, через S_1, на который надеть — через S_K, а вспомогательный стержень через S_W.

Оформим алгоритм решения задачи о переносе башни из n дисков с S_1 на S_K в виде процедуры MOVE с 4-мя параметрами: N, S1, SW, SK; алгоритм для n = 1 выделим в отдельную процедуру STEP, которая «перемещает» один диск со стержня S1 на SK.

Приведем блок-схему задачи о Ханойской башне (рис. 21).

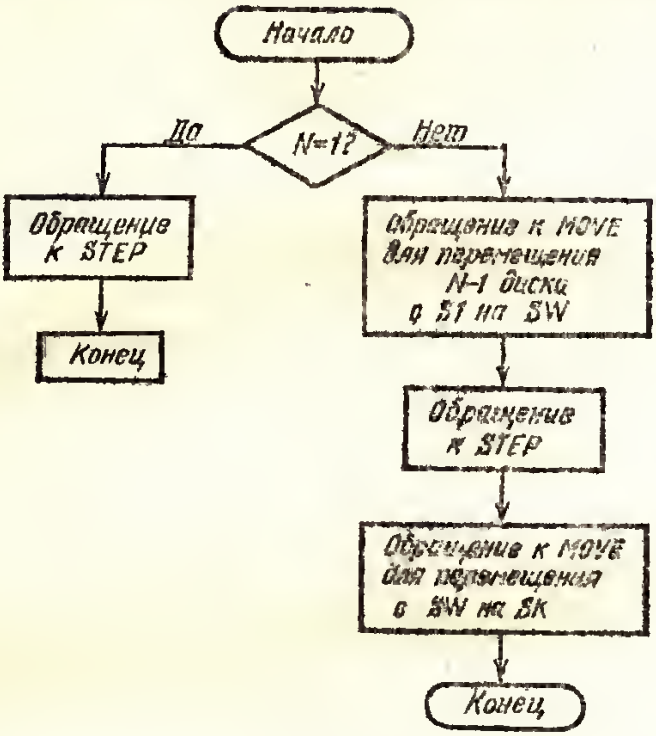


Рис. 21

Ниже приводится программа задачи о Ханойской башне и результат для перемещения 5 дисков.

```
PROGRAM HANOY (INPUT, OUTPUT);
(* ЗАДАЧА О ХАНОЙСКОЙ БАШНЕ *)
TYPE ST=(LEFT, MIDDLE, RIGHT);
(* NAT — МНОЖЕСТВО НАТУРАЛЬНЫХ ЧИСЕЛ *)
NAT=1..MAXINT;
VAR M:NAT; (* M — ЧИСЛО ДИСКОВ *)
PROCEDURE MOVE (N:NAT; S1, SW, SK:ST);
(* ПЕРЕМЕЩЕНИЕ N ДИСКОВ С S1 НА SK;
SW — ВСПОМОГАТЕЛЬНЫЙ СТЕРЖЕНЬ *)
  PROCEDURE STEP;
(* ПЕРЕМЕЩЕНИЕ ОДНОГО ДИСКА С S1 НА SK *)
    PROCEDURE PRINT (S:ST);
    BEGIN
      CASE S OF
        LEFT:WRITE ('ЛЕВ.');
        MIDDLE:WRITE ('СРЕДН.');
        RIGHT:WRITE ('ПРАВ.')
      END (* CASE *)
    END; (* PRINT *)
```

```
    BEGIN (* STEP *)
      WRITE ('СНЯТЬ ДИСК С');
      PRINT(S1);
      WRITE(' НАДЕТЬ НА ');
      PRINT(SK);
      WRITELN
    END; (* STEP *)
  BEGIN (* MOVE *)
    IF N=1 THEN STEP
         ELSE BEGIN
           MOVE (N—1, S1, SK, SW);
           STEP;
           MOVE (N—1, SW, S1, SK)
           END
  END; (* MOVE *)
  BEGIN (* HANOY *)
    READ(M); (* M — ЧИСЛО ДИСКОВ *)
    WRITELN; ('ДЛЯ', M:3, 'ДИСКОВ СЛЕДУЕТ',
         'ПРОИЗВЕСТИ СЛЕДУЮЩИЕ ДЕЙСТВИЯ:')
    MOVE (M,LEFT,MIDDLE,RIGHT)
  END.
```

Результат работы программы:

```
5
ДЛЯ 5 ДИСКОВ СЛЕДУЕТ ПРОИЗВЕСТИ
СЛЕДУЮЩИЕ ДЕЙСТВИЯ:
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
```

СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ПРАВ. НАДЕТЬ НА СРЕДН.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ЛЕВ.
СНЯТЬ ДИСК С СРЕДН. НАДЕТЬ НА ПРАВ.
СНЯТЬ ДИСК С ЛЕВ. НАДЕТЬ НА ПРАВ.

Итак, при решении поставленной задачи для пирамиды из пяти дисков необходимо произвести 31 перемещение. Это вселяет в нас оптимизм: жрецам придется долго трудиться, прежде чем «наступит конец света».

ПРИЛОЖЕНИЕ 3

НАЧИНАЮЩЕМУ ПОЛЬЗОВАТЕЛЮ ПЕРСОНАЛЬНОГО КОМПЬЮТЕРА О TURBO-ПАСКАЛЕ

Как уже упоминалось выше, трансляторы с языка паскаль имеются практически на ЭВМ всех типов. Особенно удобна для работы система Turbo Pascal на персональных компьютерах (ПК), совместимых с IBC PC (ЕС-1840/41, ЕС-1850 и др.). Эта система состоит из транслятора, сервисных и прикладных программ. Транслятор реализует стандарт языка паскаль и предоставляет ряд дополнительных возможностей, в частности, работу с данными типа BYTE (последовательность бит) и STRING (строка символов). Разрешается также использовать и прописные, и строчные буквы в любом сочетании.

В этом кратком приложении мы не ставим себе цель дать последовательное и полное описание языка паскаль для ПК. Мы хотим помочь пользователю преодолеть психологический барьер и начать работать в системе Turbo Pascal; удобный и мощный язык, комфортный сервис экранного редактора, богатые графические возможности не могут никого оставить равнодушным. Получив начальные навыки, пользователь затем в процессе работы сам освоит все необходимые ему возможности системы.

Здесь мы приводим краткое руководство по работе на ПК, иллюстрируемое на конкретных примерах. Предполагается, что пользователь знаком с клавиатурой ПК и имеет хотя бы небольшой опыт работы в системе DOS *).

Чтобы отличать информацию, выводимую компьютером на экран, от информации, набираемой пользователем, последнюю будем выделять курсивом. Название не алфавитно-цифровых клавиш возьмем в угловые скобки. Например, клавишу перевода строки будем обозначать ⟨RETURN⟩ (⟨ENTER⟩).

Пусть мы хотим составить и выполнить программу, которая вводит с клавиатуры два вещественных числа и выводит их на экран. Вот ее текст.

```
PROGRAM N1 (INPUT, OUTPUT);
VAR A,B:REAL;
```

---

*) *Пул Л.* Работа на персональном компьютере / Пер. с англ.— М.: Мир, 1986.— 383 с.

Хаузер Д., Хирт Дж., Хоукинс Б. Операционная система MS-DOS: Популярное руководство / Пер. с англ. и дополн. А. Б. Пандре.— М.: Финансы и статистика, 1987.— 168 с.

```
BEGIN
  READ (A,B); WRITELN (' A=',A,' B=',B);
END.
```

Пример сеанса работы. Вы должны иметь две дискеты: одну с системой DOS, другую с системой Turbo.

1. Дискету с DOS вставьте в дисковод A, дискету с Turbo — в дисковод B.

2. Включите компьютер. Система DOS попросит набрать текущую дату. Наберите ее, нажмите клавишу ⟨*RETURN*⟩ (⟨*ENTER*⟩). Затем в ответ на запрос системы наберите текущее время и нажмите ⟨*RETURN*⟩ (⟨*ENTER*⟩). В дальнейшем будем обозначать эту клавишу только ⟨*RETURN*⟩.

3. После загрузки DOS проведите следующий диалог:

а) A > *B*: ⟨*RETURN*⟩ — переключение на дисковод B;

б) B > *TURBO* ⟨*RETURN*⟩ — вызов системы Turbo;

в) INCLUDE ERROR MESSAGE (Y/N)? *Y* — включение выдачи диагностики ошибок;

г) >*W* — заказ рабочего файла;

д) WORK FILE NAME: *TEST1* ⟨*RETURN*⟩ — присвоение файлу имени TEST1;

е) LOADING B: TEST1.PAS

NEW FILE — сообщение системы: «новый файл»;

ж) >*E* — перевод системы в режим редактирования (EDIT)

4. После этого сверху появится строка

LINE1 COLI INSERT IDENT B: TEST1.PAS

Система расширила имя файла, добавив точку и PAS, указывающие, что файл содержит паскаль-программу. Теперь можно набирать текст программы.

```
PROGRAM N1 (INPUT, OUTPUT); ⟨RETURN⟩
VAR A,B:REAL; ⟨RETURN⟩
BEGIN ⟨RETURN⟩
  READ (A,B); WRITELN('A=',A,'B=',B) ⟨RETURN⟩
END. ⟨RETURN⟩
```

5. Если ошибок в программе вы не нашли, продолжайте диалог:

з) ⟨*F10*⟩ — выход в систему Turbo;

и) >*S* — запись файла с программой на дискету;

к) SAVING B: TEST1.PAS — подтверждение системы, что запись выполнена;

л) >*R* — запуск задачи на трансляцию и счет.

Если система не обнаружит ошибок в программе, на экране появится слово RUNNING.

м) В ответ наберите через пробел два вводимых вещественных числа, например,

5.2 3.8 ⟨RETURN⟩

На экран выдается результат:

A=5.2000000000E + 00 B=3.8000000000E + 00

6. Пусть в программе были допущены ошибки, например,

```
REAAD (A,B); WITELN(' A=',A,' B=',B);
```

Тогда по команде *R* во время трансляции будет выдана диагностика в нижнюю строку экрана:

ERROR 41:Unknown identifier or syntax error. Press ⟨ESC⟩.

7. Нажмите ⟨*ESC*⟩. На экране появится текст программы, и мигающий курсор укажет на место первой ошибки (установится в начале слова READD).

8. Для устранения ошибок воспользуйтесь средствами экранного редактора. Он не требует никаких команд для своего вызова и всегда готов к работе.

В приведенном примере необходимо выполнить два действия: стереть лишнюю букву A в READD и вставить R в WITELN.

Вставка символа. Подведите курсор к нужной позиции (под букву I) и нажмите вставляемый символ (*R*). Вся строка, начиная с указанного места, сместится вправо, а вставляемый символ встанет на место.

Удаление символа. Подведите курсор под удаляемый символ и одновременно нажмите две клавиши: ⟨*CTRL*⟩ и *G*. Символ будет удален, а строка сомкнется.

В конце приложения приводится перечень команд экранного редактора системы Turbo.

9. После исправления ошибок нажмите клавишу ⟨*F10*⟩, затем *S*. Файл с откорректированным текстом программы запишется под прежним именем (TEST1.PAS), а предыдущий файл также будет сохранен на дискете, но расширение PAS система заменит на BAK (TEST1.BAK).

10. Чтобы посмотреть, какие файлы хранятся на дискете, достаточно набрать

```
>D
DIR MASK: ⟨RETURN⟩
```

11. Для выхода из системы Turbo в DOS служит команда

```
>Q
```

Наши первые программы. Эти несложные программы помогут пользователю персонального компьютера на практике познакомиться с некоторыми возможностями системы Turbo.

Программа 1. Разбиение вводимой строки на слова

```
PROGRAM KOSYANIN (INPUT,OUTPUT);
VAR S1,S2: STRING[100]; I:INTEGER;
BEGIN
  WRITELN (' ВВЕДИТЕ ТЕКСТ ');
  READLN(S1); I:=1;
  WHILE I < LENGTH(S1) DO
    BEGIN
      S2:='';
      REPEAT
        S2:=S2 + S1[I]; I:=SUCC(I)
      UNTIL (S1(I)='␣') OR (I > LENGTH(S1));
      WRITELN(' СЛОВО — ',S2)
    END
END.
```

Эта программа иллюстрирует некоторые возможности работы с переменными типа STRING (строка символов). Здесь стандартная функция LENGTH(S1) дает длину строки S1, т. е. число символов в ней; оператор S2:='' означает, что в S2 заносится «пустая» строка, не содержащая ни одного символа; операция + «сцепляет» две строки в одну.

Пример диалога.

ВВЕДИТЕ ТЕКСТ
ПРОГРАММА БЫЛА НЕБОЛЬШОЙ.
Результат работы программы
СЛОВО — ПРОГРАММА
СЛОВО — БЫЛА
СЛОВО — НЕБОЛЬШОЙ.

Программа 2. Преобразование даты

```
PROGRAM KRYUKOV (INPUT,OUTPUT);
LABEL 1;
TYPE STR10 = STRING[10];
VAR DAY,YEAR,YEAR1, MONTH,
  I,J,K,L,M: INTEGER;
  MONTH1: STR10;
CONST MONTHES:ARRAY [1..12] OF STR10 =
  ('ЯНВАРЯ ','ФЕВРАЛЯ ', 'МАРТА ',
  'АПРЕЛЯ ','МАЯ ','ИЮНЯ ',
  'ИЮЛЯ ','АВГУСТА ','СЕНТЯБРЯ ',
  'ОКТЯБРЯ ','НОЯБРЯ ','ДЕКАБРЯ ');
BEGIN
  WRITELN (' ВВЕДИТЕ ДАТУ СВОЕГО РОЖДЕНИЯ ');
  WRITELN (' ПО ФОРМАТУ DD MM YY');
  READ (DAY, MONTH, YEAR);
  {ПРОВЕРКА ПРАВИЛЬНОСТИ ДАТ}
  WHILE (DAY < 1) OR (DAY > 31) OR
     (MONTH < 1) OR (MONTH > 12) OR
     (YEAR < 0) OR (YEAR > 87) DO
     BEGIN
       WRITELN (' ОШИБКА ВВОДА'); GOTO 1;
     END;
    YEAR1:=1900 + YEAR;
    MONTH1:=MONTHES [MONTH];
    WRITE(' ВАШ ДЕНЬ РОЖДЕНИЯ: ');
    WRITE(' ', DAY,' ', MONTH1,' ', YEAR1,';
1:END.
```

Пример диалога.

ВВЕДИТЕ ДАТУ СВОЕГО РОЖДЕНИЯ
ПО ФОРМАТУ DD MM YY
03 10 71
ВАШ ДЕНЬ РОЖДЕНИЯ: 3 ОКТЯБРЯ 1971 ГОДА

В программе использован массив MONTHES, состоящий из переменных-строк, описанный в разделе CONST.

Программа 3. Орнамент

```
PROGRAM KOSYANIN(INPUT, OUTPUT);
{$I GRAPH.P}
VAR I,J,K,L,M,N: INTEGER;
BEGIN
  GRAPHCOLORMODE;
```

```
    WRITELN(' ВВЕДИТЕ КООРДИНАТЫ ЦЕНТРА ');
    WRITELN(' X: ОТ 0 ДО 320; Y: ОТ 0 ДО 200 ');
    WRITE('X='); READ(I); WRITELN;
    WRITE('Y='); READ(J); WRITELN;
    WRITELN (' УКАЖИТЕ НОМЕР ЦВЕТА ДЛЯ ФОНА ');
    WRITELN (' ОТ 0 ДО 15 ');
    WRITE (' N='); READ(N); WRITELN;
    WRITELN (' УКАЖИТЕ НОМЕР ЦВЕТА
                ДЛЯ ОРНАМЕНТА ');
    WRITELN (' ОТ 0 ДО 3 ');
    WRITE (' M='); READ(M);
    CLEARSCREEN; GRAPHBACKGROUND(M);
    FOR K:=1 TO 61 DO
      BEGIN
        ARC (I,J,360,23,M); TURNLEFT(6);
      END
  END.
```

Эта программа рисует орнамент с заданным центром. Вводятся также цвет орнамента и цвет фона. Необходимые данные программа запрашивает в процессе выполнения.

Программа использует графический пакет GRAPH.P, который описывается как внешний модуль посредством псевдокомментария {$I GRAPH.P}. В общем виде внешний модуль с именем F описывается как {$I_F}.

Опишем некоторые стандартные процедуры для работы с графикой.

GRAPHCOLORMODE — переводит экран в графический режим. Координаты точек на экране задаются парой чисел (X,Y), где $0 \leqslant X \leqslant 319$, $0 \leqslant Y \leqslant 199$.

GRAPHBACKGROUND(N) — закрашивает весь экран цветом с номером N, где $0 \leqslant N \leqslant 15$. Соответствие параметра N и цвета фона следующее: 0 — черный; 1 — синий; 2 — зеленый; 3 — голубой; 4 — красный; 5 — малиновый; 6 — коричневый; 7 — светло-серый; 8 — темно-серый; 9 — светлосиний; 10 — светло-зеленый; 11 — светло-голубой; 12 — светло-красный; 13 — розовый; 14 — желтый; 15 — белый.

Это соответствие справедливо, однако, не для всех мониторов. Если монитор вообще не цветной, то смена цвета приведет лишь к изменению яркости.

ARC (X,Y,ALFA,R,M) — рисует дугу, начиная от точки (X,Y), размером ALFA градусов, радиусом R, цвета M. Если $ALFA > 0$, то дуга рисуется по часовой стрелке; если $ALFA < 0$ — против часовой стрелки. M = 0;1;2;3.

PLOT (X, Y, M) — ставит точку цвета M с координатами (X, Y). M = 0;1;2;3.

DRAW (X1,Y1,X2,Y2,M) — рисует отрезок цвета M, соединяющий точки (X1,Y1) и (X2,Y2). M = 0;1;2;3.

CIRCLE (X,Y,R,M) — рисует окружность цвета M с центром в (X,Y) и радиусом R.

GetPic (A,X1,Y1,X2,Y2) — копирует в массив A(буфер) участок экрана — прямоугольник с координатами концов его диагонали (X1,Y1) и (X2,Y2).

Минимальный размер NMIN буфера в байтах следующий:
для экрана 320 × 200 точек

NMIN = ((| X1 — X2 | + 1)DIV 4) 2(| Y1 — Y2 | + 1) +6;
для экрана 640 × 200 точек
NMIN = ((| X1 — X2 | + 7) DIV 8) (| Y1 — Y2 | + 1) + 6.

Во втором и третьем байте буфера A после выполнения GETPIC (A,X1,Y1,X2,Y2) запоминается высота и ширина копируемого прямоугольника.

PutPic (A,X,Y) — копирует содержимое буфера A на экран в прямоугольник, у которого нижняя левая вершина имеет координаты (X,Y). В буфере A предварительно должна быть запомнена картинка (цвет каждой точки внутри прямоугольника).

TurnLeft (ALFA) — поворот направления рисования на угол ALFA градусов. Если ALFA > 0, то поворот против часовой стрелки, ALFA < 0 — по часовой стрелке.

Программа 4. Управляемый объект.

```
PROGRAM KRYUKOV (INPUT,OUTPUT);
{$I GRAPH.P}
VAR C,A,B:ARRAY [1..200] OF BYTE;
  I,J,X,Y,CONT:INTEGER;
  EOJ:BOOLEAN;CH,CH1:CHAR;
BEGIN
  {ОСТАНОВ — НАЖАТИЕ КЛАВИШИ <ПРОБЕЛ>;
  ПУСК — НАЖАТИЕ КЛАВИШИ
         НАПРАВЛЕНИЯ (↑,→,↓,←);
  ИЗМЕНЕНИЕ РЕЖИМА: СЛЕД — ДА/НЕТ — D;
  СТИРАНИЕ ИЗОБРАЖЕНИЯ — C;
  ВОЗВРАТ В СИСТЕМУ TURBO — E}
  GRAPHCOLORMODE;
  DRAW (1,1,5,5,3); DRAW (5,1,1,5,2);
  GETRIC (A,0,0,6,6); GETPIC (B,1,1,5,5);
  CLEARSCREEN;
  I:=160; J:=100; PUTPIC(A,I,J);
  X:=0; Y:=0; EOJ:=FALSE; CONT:=2;
  WHILE NOT EOJ DO
    BEGIN {WHILE}
      IF KEYPRESSED THEN
        BEGIN {KEYPRESSED}
          READ (KBD,CH);
          IF(CH=#27)AND KEYPRESSED
                THEN READ (KBD, CH);
          CASE CH OF
            #72: BEGIN X:=0;Y:=-1 END;
            #75: BEGIN X:=-1;Y:=0 END;
            #77: BEGIN X:=1;Y:=0 END;
            #80: BEGIN X:=0;Y:=1 END;
            #71: BEGIN X:=-1;Y:=-1 END;
            #73: BEGIN X:=1;Y:=-1 END;
            #79: BEGIN X:=-1;Y:=1 END;
            #81: BEGIN X:=1;Y:=1 END;
            #32: BEGIN X:=0;Y:=0 END;
            #101: BEGIN EOJ:=TRUE END;
            #100: BEGIN CONT:=CONT+1 END;
            #99: CLEARSCREEN
          END {CASE}
        END; {KEYPRESSED}
      IF (CONT MOD 2)=0
```

```
            THEN C:=A ELSE C:=B;
            I:=I + X;J:=J + Y;
            {'ЗАБОР' ПО ПЕРИМЕТРУ}
            IF I < 0 THEN I:=0;
            IF I > 312 THEN I:=312;
            IF J < 8 THEN J:=8;
            IF J > 192 THEN J:=192;
            PUTPIC (C,I,J);
        END {WHILE}
    END.
```

Эта программа создает управляемый объект («крестик»). Управление осуществляется клавишами: ↑, ↓, →, ←, ⟨*PgUP*⟩, ⟨*End*⟩, ⟨*Home*⟩, ⟨*PgDn*⟩ и ⟨*пробел*⟩ (остановка). Объект может передвигаться по экрану, не оставляя следа, а может рисовать след, подобно мелу на доске. Переключение режима «след — да/нет» производится клавишей *D*. При движении по горизонтали «крестик» оставляет след в виде двух цветных линий, по диагонали — трех линий. Экран можно стереть клавишей *C*.

Для выхода из программы в систему Turbo следует нажать клавишу *E*.

Переменные KEYPRESSED и KBD известны системе и не требуют описания в программе.

KEYPRESSED имеет тип BOOLEAN и принимает значение TRUE, если нажата какая-либо клавиша на клавиатуре.

KBD — файл, куда попадает код нажатой клавиши.

Оператор READ(KBD,CH) читает из KBD код нажатой клавиши и засылает в переменную CH. Анализируя значение этого кода, программа передвигает объект на шаг в одном из направлений: вверх, вниз, влево, вправо, по биссектрисам координатных углов. Коды предваряются знаком #. Если объект пытается выйти за пределы экрана, программа ставит ему ограничения. Кодировка клавиш приведена в табл. 1.

При нажатии функциональных клавиш и клавиш с ALT вырабатываются два кода, из которых первым всегда является 27. Поэтому при чтении символа из KBD следует сделать проверку на 27; если «да», то читать второй код и его анализировать. Для клавиши ⟨ESC⟩ второй код отсутствует.

Таблица 1

Таблица кодов клавиш клавиатуры персонального компьютера (совместимого с IBM PC)

Клавиша	Код	SHIFT	CTRL	ALT
F1	27 59	27 84	27 94	27 104
F2	27 60	27 85	27 95	27 105
F3	27 61	27 86	27 96	27 106
F4	27 62	27 87	27 97	27 107
F5	27 63	27 88	27 98	27 108
F6	27 64	27 89	27 99	27 109
F7	27 65	27 90	27 100	27 110
F8	27 66	27 91	27 101	27 111

Клавиша	Код	SHIFT	CTRL	ALT
F9	27 67	27 92	27 102	27 112
F10	27 68	27 93	27 103	27 113
←	27 75	52	27 115	27 178
→	27 77	54	27 116	27 180
↑	27 72	56	27 160	27 175
↓	27 80	50	27 164	27 183
HOME	27 71	55	—	27 174
END	27 79	49	27 117	27 182
PGUP	27 73	57	27 132	27 176
PGDN	27 81	51	27 118	27 184
INS	27 82	48	27 165	27 185
DEL	27 83	46	27 166	27 186
ESC	27	27	27	—
BACKSP	8	8	127	
TAB	9	27 15	—	—
RETURN	13	13	10	—
A	97	65	1	27 30
B	98	66	2	27 48
C	99	67	3	27 46
D	100	68	4	27 32
E	101	69	5	27 18
F	102	70	6	27 33
G	103	71	7	27 34
H	104	72	8	27 35
I	105	73	9	27 23
J	106	74	10	27 36
K	107	75	11	27 37
L	108	76	12	27 38
M	109	77	13	27 50
N	110	78	14	27 49
O	111	79	15	27 24
P	112	80	16	27 25
Q	113	81	17	27 16
R	114	82	18	27 19
S	115	83	19	27 31
T	116	84	20	27 20
U	117	85	21	27 22
V	118	86	22	27 47
W	119	87	23	27 17
X	120	88	24	27 45
Y	121	89	25	27 21
Z	122	90	26	27 44
[	91	123	27	—
\	92	124	28	—
]	93	125	29	—
'	96	126	—	—
0	48	41	—	27 129
1	49	33	—	27 120
2	50	64	27 3	27 121

Продолжение

Клавиша	Код	SHIFT	CTRL	ALT
3	51	35	—	27 122
4	52	36	—	27 123
5	53	37	—	27 124
6	54	94	30	27 125
7	55	38	—	27 126
8	56	42	—	27 127
9	57	40	—	27 128
*	42	—	27 114	—
+	43	43	—	—
—	45	95	31	27 130
=	61	43	—	27 131
,	44	60	—	—
/	47	63	—	—
;	59	58	—	—

Команды экранного редактора

1: На символ влево CTRL-S либо ←.
3: На символ вправо CTRL-D либо →
4: На слово влево CTRL-A либо CTRL ←.
5: На слово вправо CTRL-F либо CTRL →.
6: На строку вверх CTRL-E либо ↑.
7: На строку вниа CTRL-X либо ↓.
8: Экран на строку вверх CTRL-W.
9: Экран на строку вниз CTRL-Z.
10: На страницу вверх CTRL-R либо PgUp.
11: На страницу вниз CTRL-C либо PgDn.
12: Влево по строке CTRL-Q CTRL-S либо Home.
13: Вправо по строке CTRL-Q CTRL-D либо End.
14: На начало страницы CTRL-Q CTRL-E либо CTRL-Home.
15: На конец страницы CTRL-Q CTRL-X либо CTRL-End.
16: На начало файла CTRL-Q CTRL-R либо CTRL-PgUp.
17: На конец файла CTRL-Q CTRL-C либо CTRL-PgDn.
18: На начало блока CTRL-Q CTRL=B.
19: На конец блока CTRL-Q CTRL-K.
20: На последнюю поаицию курсора CTRL-Q CTRL-P.
21: Режим вставка/замена CTRL-V или INS.
22: Вставка строки CTRL-N.
23: Стирание целой строки CTRL-Y.
24: Стирание до конца строки CTRL-Q CTRL-Y.
25: Стирание слова справа от курсора CTRL-T.
26: Стирание символа CTRL-G.
27: Стирание символа слева от курсора DEL.
29: Метка «начало блока» CTRL-K CTRL-B или F7.
30: Метка «конец блока» CTRL-K CTRL-K или F8.
31: Метка одиночного слова CTRL-K CTRL-T.
32: Появление/исчеановение блока на экране CTRL-K CTRL-H.
33: Копирование блока CTRL-K CTRL-C,

34: Перепись блока CTRL-K CTRL-V.
35: Стирание блока CTRL-K CTRL-Y.
36: Чтение блока с диска CTRL-K CTRL-R.
37: Запись блока на диск CTRL-K CTRL-W.
38: Выход из редактора CTRL-K CTRL-D или F10.
41: Запоминание строки CTRL-Q CTRL-L.
42: Поиск фрагмента (фрагмент указывается в процессе диалога) CTRL-Q CTRL-F.
43: Поиск/замена фрагмента CTRL-Q CTRL-A.
Параметры:
U — игнорировать отличия прописных и строчных букв;
W — найти слово целиком;
B — двигаться в обратном направлении;
G — искать заданный фрагмент во всем файле; при нахождении фрагмента система каждый раз переспрашивает: «Заменять — да/нет?»;
N — система сама без переспрашивания заменяет все найденные одинаковые фрагменты.
Параметры можно задавать подряд в любом порядке.
44: Повторение последней команды поиска CTRL-L.
45: Признак управляющего символа CTRL-P.
Любую команду редактора можно прервать клавишей ⟨ESC⟩.

СПИСОК ЛИТЕРАТУРЫ

1. Абрамов С. А., Зима Е. В. Начала программирования на языке паскаль.— М.: Наука, 1987.—112 с., ил..
2. Вирт Н. Алгоритмы + структуры данных-программы / Пер. с англ.; Под ред. Д. Б. Подшивалова.— М.: Мир, 1985.— 406 с., ил.
3. Григас Г. Начала программирования: Кн. для учащихся / Пер. с литовского; Под ред. Ю. А. Первина. — М.: Просвещение, 1987.— 112 с., ил.
4. Йенсен К., Вирт Н. Паскаль. Руководство для пользователя и описание языка / Пер. с англ., предисл. и послесл. Д. Б. Подшивалова. — М.: Финансы и статистика, 1982.— 151 с., ил.
5. Керниган Б., Плоджер Ф. Инструментальные средства программирования на языке паскаль / Пер. с англ.; Под ред. Б. А. Ашкинази.— М.: Радио и связь, 1985.— 313 с., ил.
6. Лебедев В. Н., Соколов А. П. Введение в систему программирования ОС ЕС.— М.: Статистика, 1978.
7. Мазный Г. Л. Программирование на БЭСМ-6 в системе «Дубна» / Под ред. Н. Н. Говоруна.— М.: Наука, 1978.— 272 с., ил.
8. Пирин С. И. О реализации компилятора паскаль на ЭВМ БЭСМ-6 // Обработка символьной информации.— М.: ВЦ АН СССР, 1978.— Вып. 4.
9. Салтыков А. И., Семашко Г. Л. Программирование для всех.— М.: Наука, 1986.— 176 с.
10. Уилсон И. Р., Эддиман А. М. Практическое введение в паскаль / Пер. с англ.; Под ред. Л. Д. Райкова.— М.: Радио и связь, 1983.— 144 с., ил.
11. Форсайт Р. Паскаль для всех / Пер. с англ.; Под ред. Ю. И. Топчеева.— М.: Машиностроение, 1986.— 288 с., ил.
12. Хьюз Дж., Мичтом Дж. Структурный подход к программированию / Пер. с англ.; Под ред. В. Ш. Кауфмана.— М.: Мир, 1980.— 280 с., ил.
13. Pascal 8000 Reference Manual OS, VS and CMS. Version 2.0, 1980.
14. Welsh J., Elder J. Introduction to Pascal — London: Prentice—Hall International Inc,, 1979.

Галина Львовна Семашко,
Альберт Иванович Салтыков

ПРОГРАММИРОВАНИЕ НА ЯЗЫКЕ ПАСКАЛЬ

Серия «Библиотечка программиста», вып. 51

Редактор *О. И. Сухова*
Художественный редактор *Г. М. Коровина*
Технический редактор *Л. В. Лихачева*
Корректоры *Л. И. Назарова, М. Н. Дронова*

ИБ № 32360

Сдано в набор 12.06.87. Подписано к печати 18.02.88. Т-04573.
Формат 84×108/32. Бумага тип. № 2. Гарнитура обыкновенная.
Печать высокая. Усл. печ. л. 6.72. Усл. кр.-отт. 6,93.
Уч.-изд. л. 7,65. Тираж 210 000 экз. Заказ № 801. Цена 50 коп.

Орденв Трудового Красного Знамени издательство «Наукв»
Глввная редакция физико-математической литературы
117071 Москва В-71, Ленинский проспект, 15

2-я типография издательства «Наукв»
121099 Москвв Г-99, Шубинский пер., 6

50 коп.